1986

# DIB on the Xerox workstation

Chien-Kuo Chuang

**Rochester Institute of Technology**
**School of Computer Science and Technology**

# DIB On The Xerox Workstation

by
Chien-Kuo Eric Chuang

A thesis, Submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by: _____
Professor John L. Ellis

_____
Professor Andrew T. Kitchen

_____
Professor Margaret M. Reek

November 25, 1986

Title of Thesis: **DIB On The Xerox Workstation**

I _____ hereby grant permission to the Wallace Memorial Library, of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 5/5/87

## ACKNOWLEDGMENTS

I take this opportunity to thank Professor John Ellis for his guidance and encouragement during the course of this thesis work. I also thank Professor Andrew Kitchen and Professor Margaret Reek for their suggestions and criticisms.

I am extremely grateful to Professor Raphael Finkel for his guidance during the implementation of this project. Without his help, this thesis would have been impossible.

Most of all I would like to thank my family, for their support and understanding during the past three years.

## ABSTRACT

**DIB - A Distributed Implementation of Backtracking** is a general-purpose package which allows applications that use tree-traversal algorithms such as backtrack and branch-and-bound to be easily implemented on a multicomputer. The application program needs to specify only the root of the recursion tree, the computation to be performed at each node, and how to generate children at each node. In addition, the application program may optionally specify how to synthesize values of tree nodes from their children's values and how to disseminate information in the tree. DIB uses a distributed algorithm, transparent to the application programmer, that can divide the problem into subproblems and dynamically allocate them to any number of machines. It can also recover from failures of machines. DIB can now run on the Xerox workstation network at Rochester Institute of Technology. Speedup is achievable for exhaustive traversal and branch-and-bound, with only a small fraction of the time is spent in communication.

## TABLE OF CONTENTS

# 1. INTRODUCTION AND BACKGROUND

## 1.1 General Description

In the next few years, professionals from many areas will have access to advanced workstations connected by networks. A network of workstations is composed of machines that each can act independently but also can cooperate with others through messages. Algorithms that take advantage of these architectures are of great importance. Such algorithms can employ workstations that are frequently idle to help solve computationally intensive problems.

It is apparent from experience that writing distributed programs is significantly more difficult than writing sequential programs. Programmers have to deal with a variety of new issues, such as synchronization, concurrency, communication protocols, and fault tolerance. In the current environment, distributed programming is left mostly to experts.

One way to make distributed programming easier is to develop library packages in specific areas. These packages will be suitable only to specific applications and will lack the generality of a programming language. Because these packages are relatively easy to use, the distributed part of a program can be transparent to the user, and programmers who are unable or unwilling to master the techniques of distributed programming can still use the full power of the workstation network.

DIB A Distributed Implementation of Backtracking is a general-purpose package which allows applications that use tree-traversal algorithms such as backtrack and branch-and-bound to be implemented on a multicomputer. It was presented by Professor Finkel and Professor Manber in the "Proceedings of the 5th International Conference on Distributed Computing Systems" and currently runs on the Crystal multicomputer at the University of Wisconsin-Madison. It is easy to use. The application program needs to specify only the root of the recursion tree, the computation to be

performed at each node, and how to generate children at each node. In addition, the application program may optionally specify how to synthesize values of tree nodes from their children's values and how to disseminate information in the tree. DIB uses a distributed algorithm, transparent to the application programmer, that can divide the problem into subproblems and dynamically allocate them to any number of machines. It can also recover from failures of machines. This thesis will implement DIB on the Xerox workstation network at Rochester Institute of Technology.

## 1.2 Description Of The General Class Of Problems

The problems that can be supported by DIB are those for which the computation is performed by traversing a tree. The tree is usually built dynamically during the traversal and each tree node gets initial data from its parent. The node may decide to "generate" several children and pass more data to them, or it may decide that it is a leaf, in which case it performs some computation and passes the outcome to its parent. Information flows down the tree as it is built and flows up the tree as subtrees complete. In addition, a node can accept new information from its parent after it is generated and can pass information to its live children. DIB may expand the frontier of the tree in an arbitrary order and may expand the same node twice (to recover from failures). However, it ensures that the information reported back to a node from its children represents exactly one instance of every descendant. The only requirement of the application for using DIB is that each subtree can be correctly searched without any knowledge of the outcome of any other search. The outcome of other searches may affect the efficiency of computation, but it must not affect the correctness.

There are basically two different types or classes of applications that can use DIB. The first class of DIB applications includes exhaustive search of a tree of combinations. In these applications the outcomes of all the leaves have to be collected. Each node is either a solution, a dead end, or an intermediate situation. In the later case, its children represent alternative ways to continue. The outcomes of the computation of all the children of a particular node may have to be combined in some way to produce the outcome for this

node. The number of solutions found can be reported up the tree and the outcome of the computation is then defined as the outcome of the root. The eight-queens problem and knights tour problems are examples of which belong to this class. These problems are described further in section 6.

A second class of applications involves global knowledge that is improved when a search reaches a leaf. This is the situation in applications which use branch and bound, where the global bound is used to prune further development of unsuccessful branches. The traveling salesman problem belongs to this class. In the application program, each node *v* computes a function, called the objective function, which depends on values computed along the path from the root to *v*, and the program intends to find the leaf with the minimum value of this function. If a lower bound on the value of the objective function in a subtree rooted at a given internal node *v* exceeds the value already attained by a leaf, there is no need to explore further. A good heuristic is to explore those children with smaller objective function values first in the hope that the other children can be pruned.

### 1.3 DIB And The Operating System

DIB's requirements from the operating system are minimal: the machines must be connected by a network that supports a message-passing mechanism and each machine must be able to communicate with all other machines directly; each machine may run at a different speed.

There is no need for a clock or a timeout mechanism for DIB. DIB does not require that incoming messages generate interrupts. Instead, incoming messages are buffered; DIB will periodically check to see if new messages have arrived.

## 2. RELATED WORK

The field of distributed algorithms is growing. Several parallel implementations of specific algorithms and general methods have been reported. Decompositions of alpha-beta search have been the subject of much effort.

Parallel-aspiration search, Baudet [2], gives the entire problem to each machine, with each one constrained to find a solution within a different window. Narrow windows speed up the search.

Mandatory-work-first search, Akl and Barnard [4], distinguishes between those parts of a subtree that must be searched and those parts whose need to be searched is contingent upon search results in other parts of the tree. By searching mandatory nodes first, the algorithm attempts to achieve as many of the cutoffs seen in the serial case as possible.

The tree-splitting algorithm was described by Finkel and Fishburn [3]. The root processor evaluates the root position. Each interior processor evaluates its assigned position by generating the successors and queuing them for parallel assignment to its slave processors. Thus a processor at level N in the processor tree always evaluates positions at level N in the lookahead tree. As an interior processor receives responses from its slaves, it narrows its window and tells working slaves about the improved window. When all successors have been evaluated (or a cutoff has occurred), the interior processor is able to compute the value of its position. Each leaf processor evaluates its assigned position with the serial alpha-beta algorithm. When a processor finishes, it reports the value computed to its master.

These tree-splitting methods require a fixed machine tree. They suffer from machines sitting idle while work is still pending in other parts of the tree. They also allocate identical amounts of processing power to large regions of the tree, even though the left-most region is most likely to yield results quickly.

Moller-Nielsen and Staunstrup [5] have experimented with many different multicomputer algorithms, including branch-and-bound, and found that good speedups are possible when most of the tree has to be traversed.

Another approach to branch-and-bound [6] is to have each processor compute one node of the search tree and then reassign work based on the current bounds and cost functions. The amount of communication needed is very high.

All of these approaches require that a logical problem structure (a tree structure) is mapped in some way onto the physical machines. Under the tree-splitting, the map is dynamic, but the machines must be arranged as a tree and the map respects tree level. In DIB, the work is initially divided arbitrarily among machines. When machine $M_i$ finishes the work it was given, it sends a "request for work" to another machine $M_j$. If $M_j$ is currently working, it divides its work and sends part of it to $M_i$; otherwise, it forwards the request. So, the DIB program allows a highly dynamic mapping between logical subproblems and machines.

## 3. THE UNDERLYING ALGORITHMS

### 3.1 Algorithm for Distribution

The algorithm is based on a depth-first search of the tree. If there is only one machine then the computation is straightforward. Suppose there are ***n*** work machines $M_1$, $M_2$, .... $M_n$, $n > 1$. Each machine $M_i$ maintains two tables, WorkGiven and WorkGotten. WorkGotten records problems that were received from other machines. $M_i$ is said to be responsible for these problems. Finished work is reported to its originator and is removed from WorkGotten. This table therefore allows DIB to pass results back up the tree. WorkGiven records problems that $M_i$ has given to other machines. It can distinguish problems given to other work machines, and problems that have been generated (or received from elsewhere) but unassigned. When an answer to one of these problems is reported, its parent is notified and the entry is removed from WorkGiven. This table allows DIB to determine which work it is responsible for is still outstanding and to redo it in case of failure.

In the simple case of only one work machine, each table will contain only one entry: the root of the tree. The nodes generated during the search will be stored temporarily in a stack, not in the tables. The tables are only used to keep track of parts of the tree that cross machine boundaries.

When machine $M_i$ has finished a problem and reported its result, it picks the next problem to attack from WorkGiven. If there are several unassigned problems there, it picks the earliest one, the one that appears first in a symmetric-order traversal of the tree. Unless it needs to subdivide this problem to give some away to another work machine, $M_i$ uses its stack to record the progress of the search. If WorkGiven has no available work, $M_i$ sends requests for work to other machines, which are selected by an algorithm that will be described later. WorkGiven might contain unavailable work, that is, work distributed to other machines.

A machine $M_j$ might receive a request for work from $M_i$ while it is engaged in a search. $M_j$ sets the search aside for a moment to try to grant this request by sending $M_i$ some work, the earliest unassigned problems in WorkGiven. If there is no available work in WorkGiven, the current problem $M_j$ is working on is subdivided, and its children are placed in WorkGiven. The child currently being searched by $M_j$ is marked as assigned to $M_j$: its later siblings are marked as unassigned. Some of these later are then sent to $M_i$. This subdivision is repeated until at least one unassigned problem is generated to send to $M_i$. However, if subdivision reaches a trivial problem, or if it reaches the depth at which $M_j$ is currently working, the request is not granted. Instead, it is forwarded to some other work machine. After dealing with $M_i$'s request, $M_j$ returns to its own search. Because the time needed to perform a piece of work and the relative speeds of the machines are not known in advance, they are not used in determining how much work to give away or which machine to ask for work. If $M_i$ receives what turns out to be the bulk of $M_j$'s work or suddenly becomes slow, other machines will finish their work and get $M_i$'s part. Thus, DIB contains an "automatic" load balancing mechanism.

The strategy used to select work to perform or to give away tends to work on the earlier parts of the search tree first. However, it does not guarantee any particular order of search because of the nondeterministic nature of work distribution.

The algorithm for sending request messages is that each machine $M_i$ sends requests for work to $k$ other machines selected at random. The constant $k >= 1$ may depend on machine numbers and on the application. If the selected machine $M_j$ can not grant the request then it forwards the request to other machine, if it is forwardable. If the request has already got "Hops = work machines" then $M_j$ will ignore the request. $M_i$ may request for work again if it has been idle some period of time. So if one machine fails, the failed machine will not stop the whole system.

## 3.2 Algorithm for Fault Tolerance

Fault tolerance is a crucial area in distributed computing. In an environment of personal workstations, hardware or software failures are not the only types of failures. For example, when an application is running on other machines, the machine's owner may decide at one point to stop all "guest" processes. As a result, the likelihood of failures may be independent of the hardware, and their distribution may be unpredictable.

The fault-tolerance algorithm works as follows. When a machine $M_i$ finishes its work and its WorkGiven table is not empty, some other machine still has work to do or it is likely that the work $M_i$ gave away went to a failing machine. Because $M_i$ can not distinguish between these two possibilities and since $M_i$ has to wait in any case for another machine to grant its request, it can use the waiting time to redo some outstanding work in its WorkGiven table. Redundant work has low priority and is discarded if $M_i$'s request is granted with non-redundant work. If $M_i$ is asked for work while computing redundant work, it grants the request as before, labelling the work it grants as redundant. Even this work may be redone by $M_i$ later, at a higher level of redundancy.

Whenever $M_i$ needs to select work from WorkGiven to perform or give away, it chooses available work at the lowest redundancy level. If there is no available work, it chooses work given away at the lowest redundancy level to redo, raising its redundancy in the process. Within a redundancy level, work earlier in the tree is chosen. Since the same work may be performed by two machines, its result may be received twice. The first result removes the entry from WorkGiven, and the second result is discarded. As long as machine 1 survives, which is responsible for the root, the tree will eventually be searched completely.

One problem called the ancestral-chain problem may happen in a chain of processes ancestral to a failing process. Assume that $M_1$ gave some work to $M_2$, which in turn gave some of it to $M_3$, which failed before reporting the

result back to $M_2$. When their other work is finished, both $M_1$ and $M_2$ will redo the work they gave away. What $M_2$ is redoing is worth the effort: $M_1$ however is redoing both work already finished but not yet reported by $M_2$ and the work being redone by $M_2$, both of which are unnecessary. This problem becomes more serious with longer chains.

One method which can be used to reduce this unnecessary computation is called "tell child". When $m_1$ starts redoing work, it informs $m_2$ of this fact, $m_2$ treats this information as a request for work and grants some of its redundant (but necessary) work to its parent $m_1$. This method often cures the ancestral-chain problem, but it still does not ensure that redundant work is well distributed.

The benefits of the fault tolerance algorithm are that strict timeouts are not needed and this mechanism is independent of the operating system.

# 4. IMPLEMENTATION

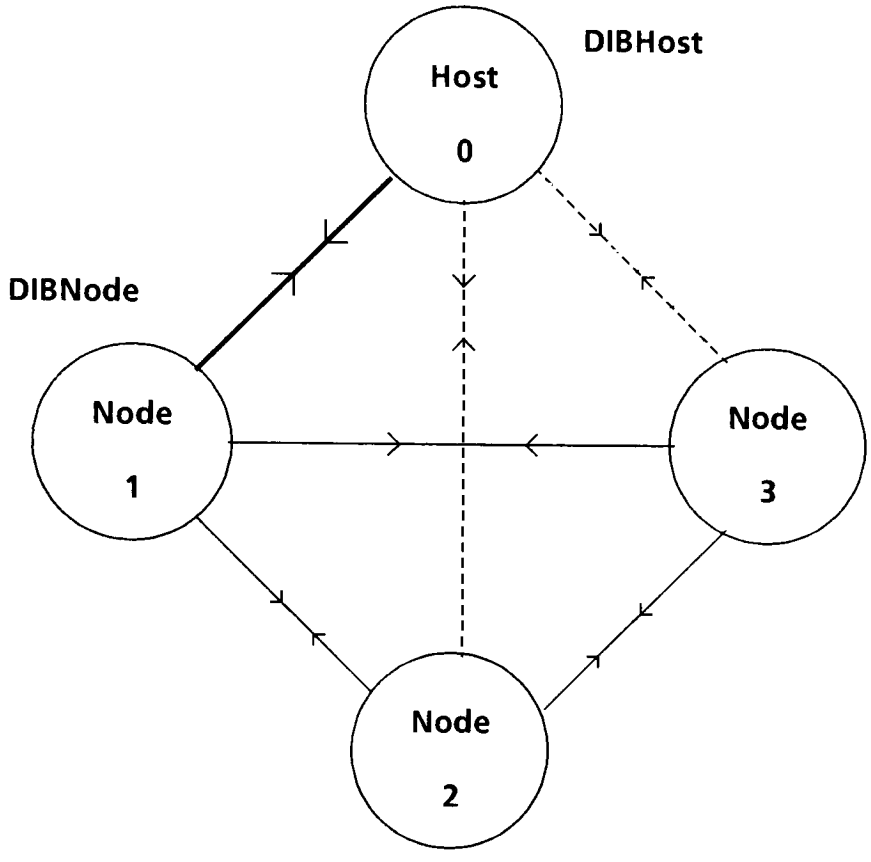
The DIB package produces two executable programs, DIBHost.bcd and DIBNode.bcd. The first program DIBHost.bcd resides on the Host (supervisor) workstation and runs interactively, asking for Node (work) workstation numbers, setting up the working environment, asking for problem sizes, collecting and printing out the execution results. The second program is placed on every Node workstation. It communicates infrequently with the Host program, principally for debugging and statistics collection, and regularly with other Node workstations. Refer to Figure 4.0 DIB's Working Environment.

## 4.1 Host Implementation

The data structure for a "message" transmitted between machines is shown on Figure 4.1 and the flow chart for the Host program is shown on Figure 4.2. The Host program works as follow.

1. Set up environment

    a. Host gets Node workstation numbers from user input.
    b. Asks user to input each workstation's name (including Host), and uses the name to find each workstation's network address.
    c. According to the workstation input order, stores the network address in an array.
    d. Sends the network address array and workstation's number, which is their location in the array, to every Node workstation.
    e. After the set up step, each Node knows his number, Host's network address, and all other workstation's number and network address.




1. DIBHost runs interactively.

2. DIBNode communicates infrequently with the Host and regularly with other Node workstations.

**Figure 4.0 DIB's Working Environment**

```
StatusSpecification: TYPE = {BatchDone,AnAnswer,Quitting,
            Reporting,PleaseStop};

MessageType: TYPE =  { Request,Work,GlobalInfo,Result,
                Terminate,Synch,Repeating,Updating};


Message: TYPE = RECORD[

   specifics: SELECT type: MessageClass FROM

    AnswerMessage = > [
     Status: StatusSpecification,
     Prob: ApplicDfs.ProblemType,
     Times: TimeType,
     Sequence: LONG INTEGER,
     AnswerDewey: DeweyDfs.Dewey],

    ProblemMessage = > [
     Kind: INTEGER, -- 0 means no work; 1 means here is a problem
     Answer: AnswerSpecification,  -- Full, Count
     Prob: ApplicDfs.ProblemType,
     PartSize: INTEGER, -- partition size
     WorkFraction: INTEGER, < <how much to give away when asked;
            0 means give away none; 10 means give away
              all work; negative means give away one
              problem (if possible) > >
     NumHelp: INTEGER, -- how many to ask when out of work
     DebugLevel: INTEGER], < <higher numbers are more verbose; 0 is
                silent > >

    InterNodeMessage = > [
     Kind: MessageType,
     Asker: INTEGER,  --node # for Request and Repeating,
                          --owner for Work
     Child: ApplicDfs.ProblemType,
     ChildDewey: DeweyDfs.Dewey,
     Count: INTEGER,  --size of work, for Work
                          --number of hops for Request
     SomeInfo: ApplicDfs.InfoType,
     Redundancy: INTEGER],

   ENDCASE];
```

**Figure 4.1 Data Structure For Message**

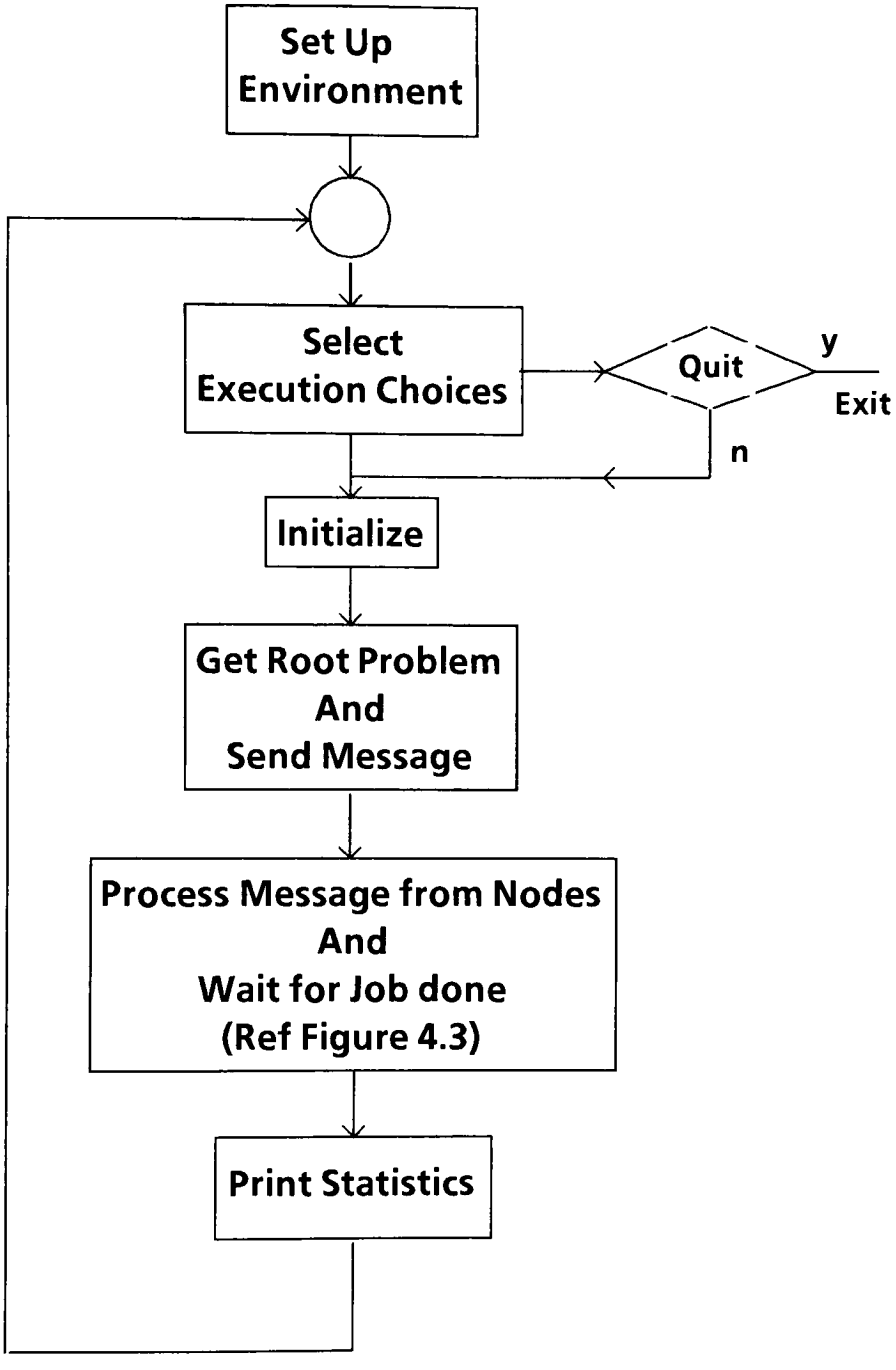



Figure 4.2 Flow Chart For Host

2. Select execution choices

   After "Set up environment", a menu will be shown on the screen. The possible execution choices are " c, f, d, h, n, q ", representing the following :

   c: count, only the final result will be reported to Host. The default value is c.

   f: full, each possible answer will be reported to Host.

   d: debug level; trace the execution of the application program. The possible value are 1,2,3, and 6. The bigger the number is, the more details of the execution will be shown on the screen. The default value is 0.

   h: helpers, number of helpers when out of work. Due to the number of workstations that can be used, helpers is set to 1 by default.

   n: new problem (run the application program with different sizes).

   q: quit.

3. After initializing the variables in the Host program and application program, the Host asks the user to input application problem sizes. Using the problem sizes as a parameter, Host calls application program to generate the "Root Problem", the problem to be solved.
   Then Host puts the root problem, the selected execution choices into a problem message package, and sends it to each Node workstation.

4. After sending the problem message package, the Host processes each message transmitted back by the Nodes, and waits for the job done message from Node 1. The flow chart for Host wait job done is shown on Figure 4.3. The message transmitted back by each Node has the following meaning.

   a. Please Stop: Node 1 tells the Host to stop.
   b. Batch Done: The whole problem has been finished, Node 1 tells the updated root to the Host.
   c. Reporting: One Node has finished a subproblem; tells the Dewey number (problem ID) of what just finished to the Host.
   d. An Answer: A result has been sent back to the Host.

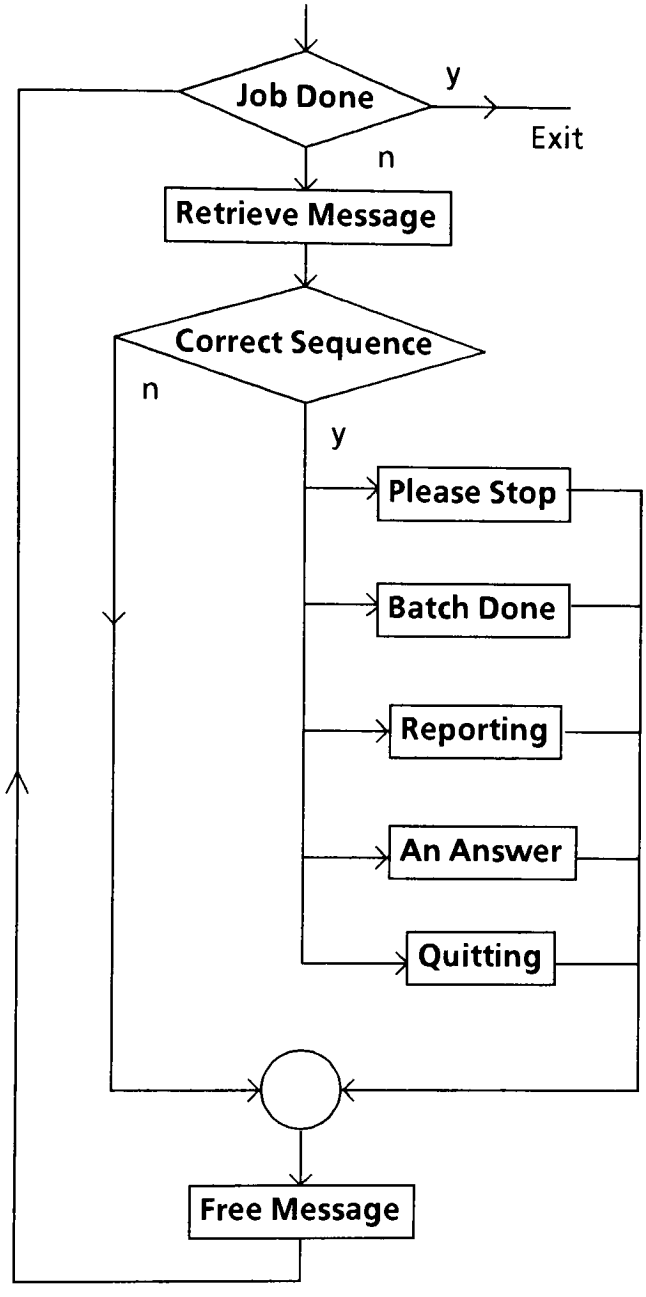



**Figure 4.3 Flow Chart For HWaitDone**

e. Quitting: Which Node is quitting.

Finally, the Host prints out the statistics results of the execution in each Node, and goes back for the next execution.

## 4.2 Node Implementation

The data structures for the WorkGotten and WorkGiven Tables is shown on Figure 4.4 and the flow chart for the Node program is shown on Figure 4.5. The Node program works as follow.

1. Node workstations will wait for the set up message from Host. After the message is received, each Node knows his number and each other workstations' number and network address. Until now, workstations could communicate with each other directly.

2. Next, Node workstations wait for the problem message from Host.

3. Using the execution choices part in the problem message, Node workstations initialize their global variables.

4. In order to let Node workstations start the execution at nearly the same time, the Node workstation with the biggest number sends start up message to the other Node workstations.

5. After synchronizing between Node workstations, Node 1 begins to work on the problem given by Host. Other Node workstations will send a "request for work" message to Node 1 to help solve the problem.

6. The algorithms for work distribution and fault tolerance have been given on previous section. The flow charts for the main procedures in the Node program are shown on Figure 4.6 BackTrack, Figure 4.7 AskForHelp, Figure 4.8 CheckIncoming, Figure 4.9 HandleRequest. The following is the descriptions of these procedures.

```
WorkGottenType: TYPE = RECORD[
    ID: DeweyDfs.Dewey,
    Owner: INTEGER, --who is waiting for the results
    UnsolvedCount: INTEGER, <<how many subproblems
                            are still under computation. >>
    Value: ApplicDfs.ProblemType,
    Next: WorkGottenPtr];


WorkGivenType: TYPE = RECORD[
    ID: DeweyDfs.Dewey,
    Value: ApplicDfs.ProblemType,
    Worker: INTEGER, --who is working on this problem for
                     --us
    Available: BOOLEAN, <<may we work on it?
                If not given to anyone, Worker = -1 (Open),
                    Available = TRUE
                If given to machine m, Worker = m,
                    Available = FALSE
                If given but ready to redo, Worker = m,
                    Available = TRUE
                If given and I am redoing, Worker = m,
                    Available = FALSE >>
    Parent: WorkGottenPtr,
    Redundancy: INTEGER,
    Next: WorkGivenPtr];
```

**Figure 4.4 Data Structure For WorkGotten And WorkGiven**

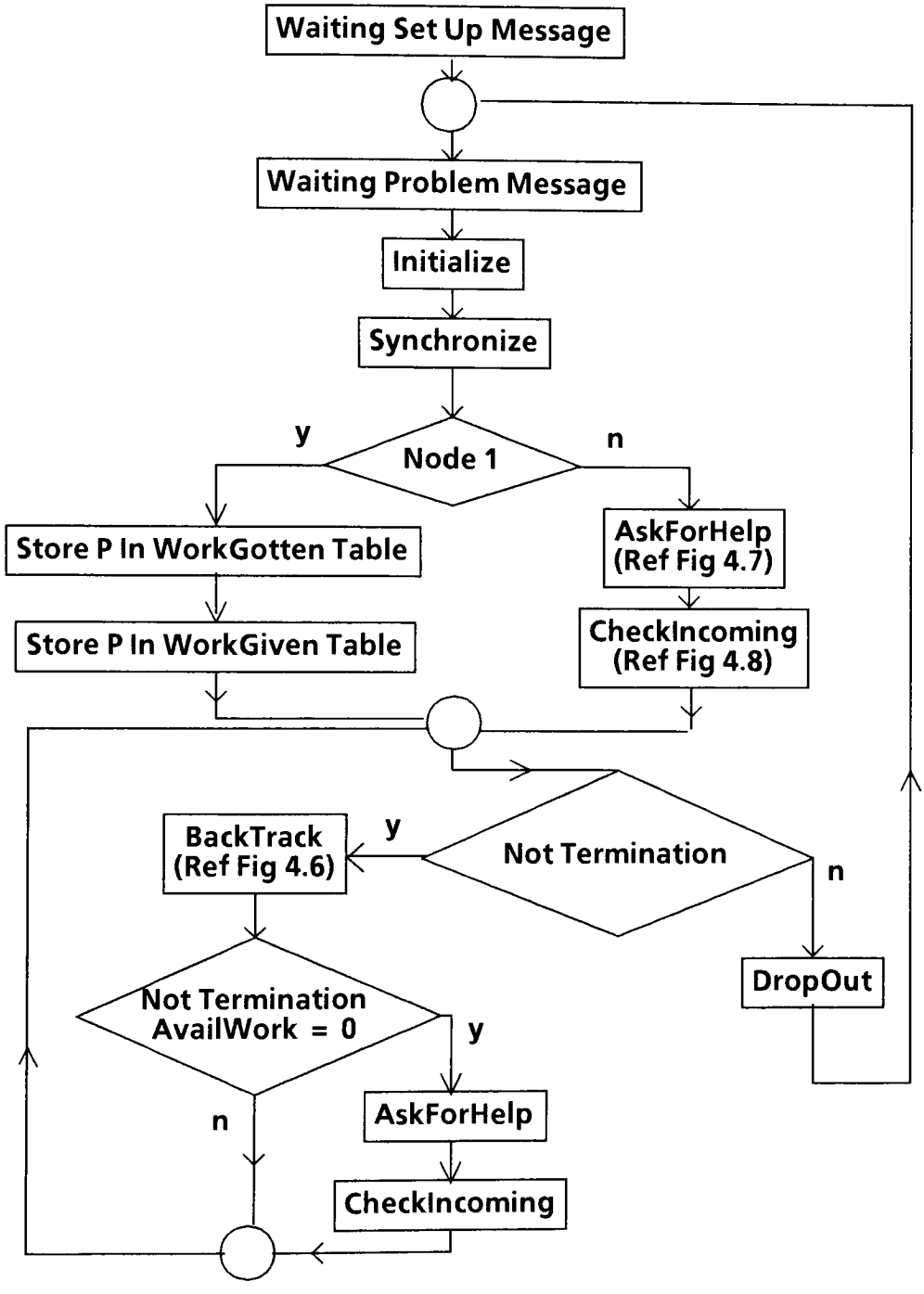



**Figure 4.5 Flow Chart For Node**

a. BackTrack: PROC[]
   Recursive backtrack to solve the problem chosen from WorkGiven table. It uses an explicit stack, called CurrentProblem, to store unfinished work. BackTrack uses the Depth-First search to find a child or a sibling of current child. Refer to Figure 4.6.1 Search in BackTrack for detail.

b. AskForHelp: PROC[FirstTime: BOOLEAN, Begger, Hops: INTEGER]
   If FirstTime then ask Node 1 for work; afterwards, send a request to other Nodes (equal to the number of helpers) chosen by random. If Begger # MyName, then forward the request. Also tell the number of hops to the helpers.

c. CheckIncoming: PROC[BusyWait: BOOLEAN]
   RETURNS[Abort: BOOLEAN]
   Called periodically to see if other Nodes have asked or forwarded a request for some work to do. If it has, call procedure HandleRequest to deal with the request. If BusyWait is set, don't return if nothing has come in; keep checking. Set Abort to true if work just came in that is less redundant than the current level. The incoming message has the following meaning.

   a. Synch: Wait for this message before continuing and set the Start time.
   b. Request: Someone is asking me for work. Call HandleRequest to deal with the request.
   c. Work: Incoming message was a response to my request, or to my notice that I am repeating work.
   d. Repeating: Someone is redoing the work that he gave to me. Try to grant him some useful work.
   e. Result: Someone has finished a problem; I need to tell each parent.
   f. Updating: The problem in the incoming message has changed, I need to update the problem.
   g. GlobalInfo: Incoming message was new global information.
   h. Terminate: The whole problem has been finished, Node 1 tells me to stop the work and reports the statistics data.

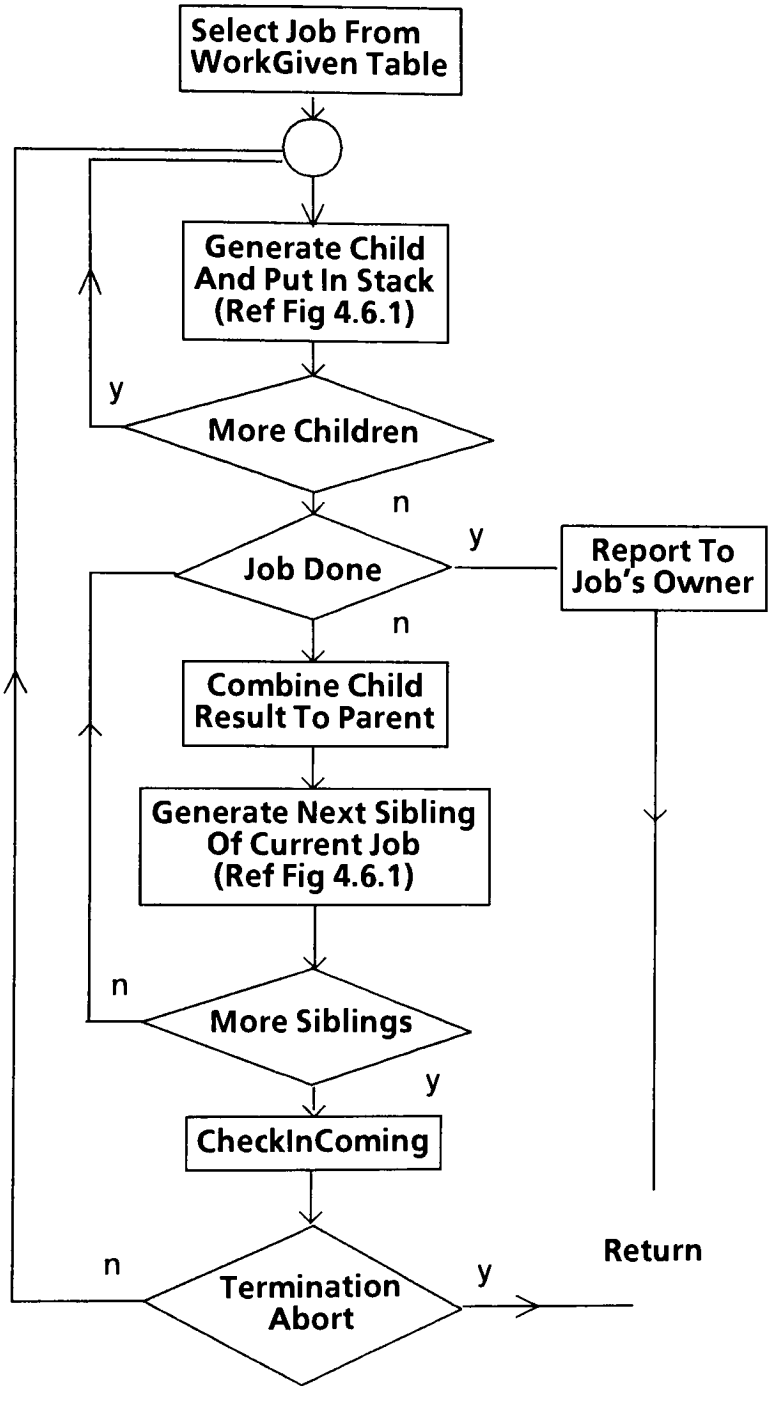



**Figure 4.6 Flow Chart For BackTrack**

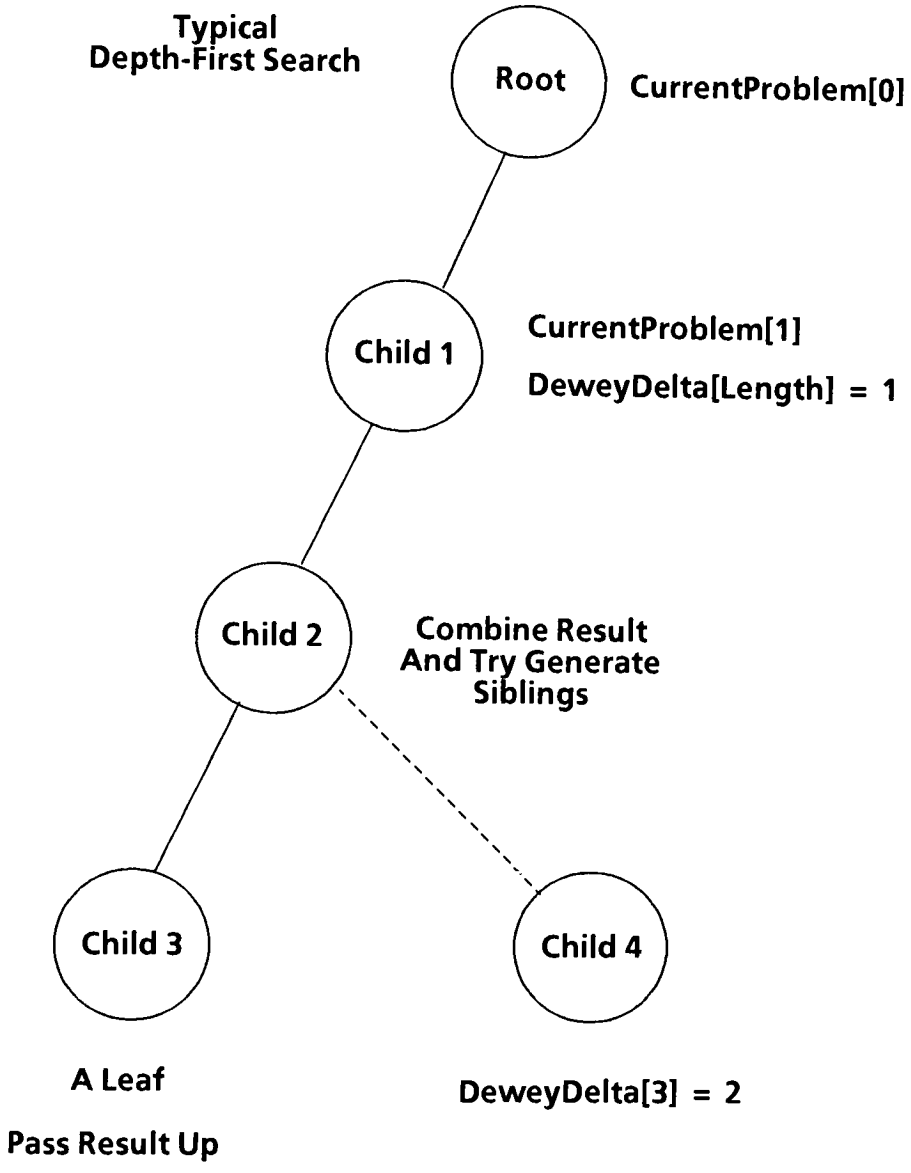



**Figure 4.6.1 Search In BackTrack**

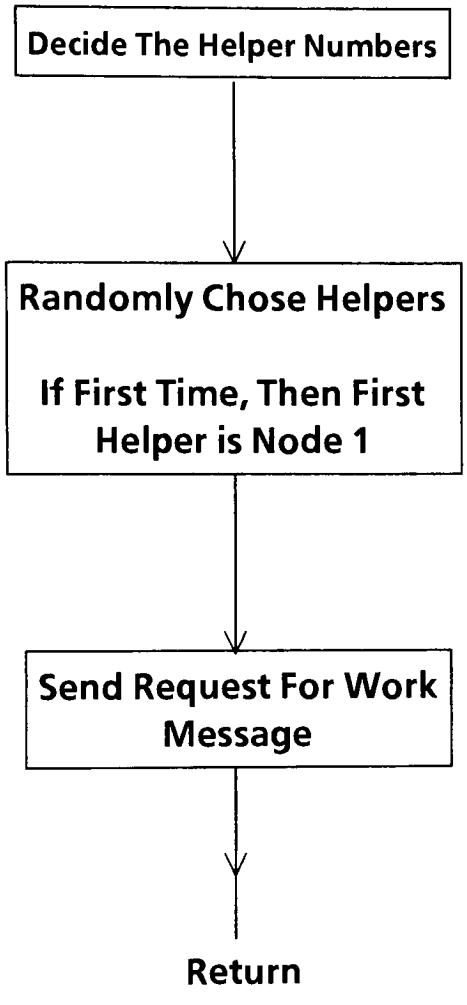



Figure 4.7 Flow Chart For AskForHelp

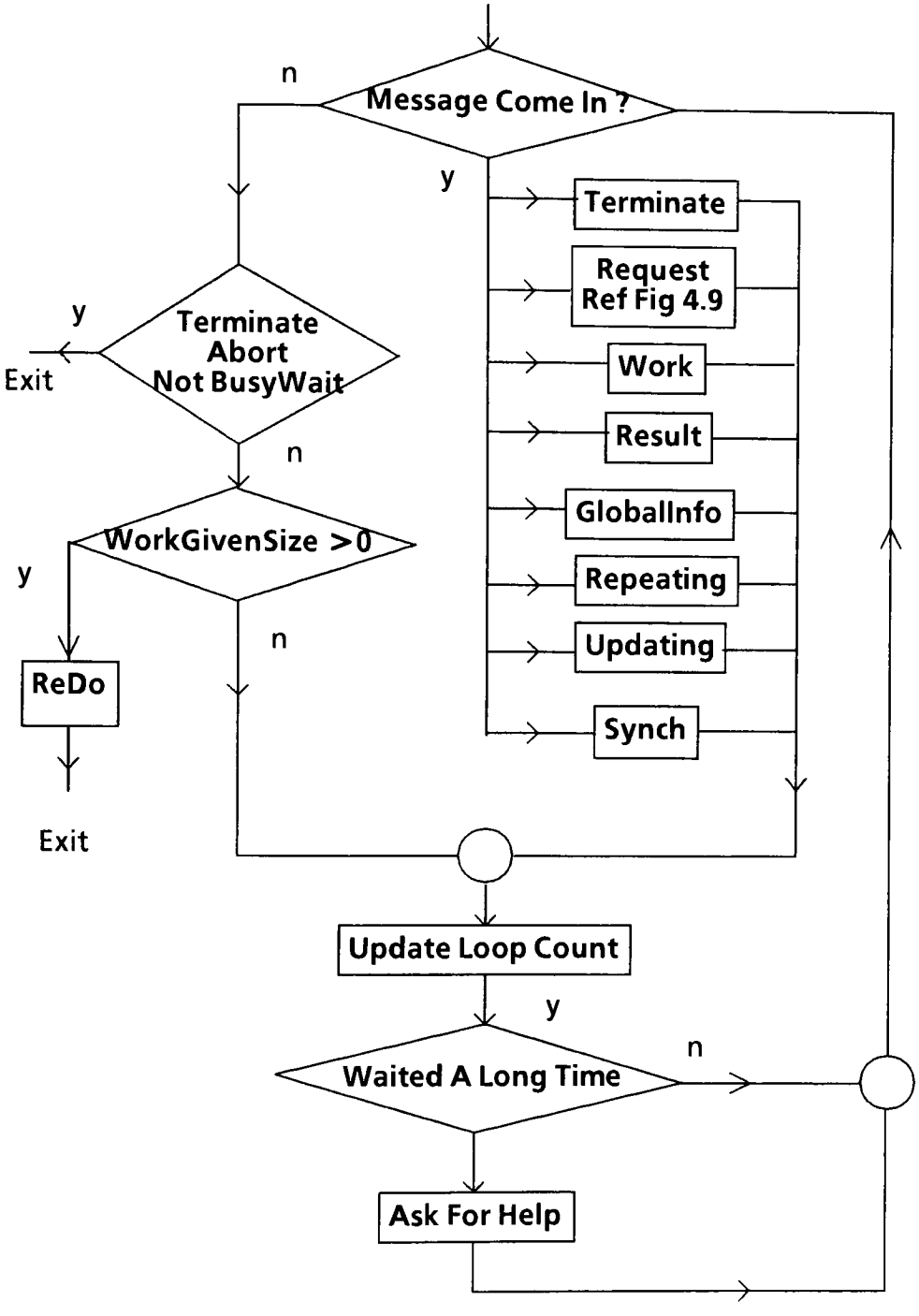



**Figure 4.8 Flow Chart For CheckIncoming**

d. HandleRequest: PROC[Who: INTEGER, Hops: INTEGER]
Called from CheckIncoming. Who has just asked us for work, or we have decide to pass some work to Who. If there is any work to spare, send it; otherwise, forward the request, if it is forwardable. If it has already got Hops = Node Machines, don't forward it. Refer to Figure 4.9.1 Subdivide Work In HandleRequest about how to subdivide work from CurrentProblem.

## 4.3 Communication Interface Implementation

The communication between workstations is based on the communication package, NetworkStream. A Network stream is the principal means by which clients of Pilot communicate between machines. NetworkStream provides access to the implementation of the Sequenced Packet Protocol -- a level 2 Internet Transport Protocol which is defined in Xerox Internet Transport Protocols. It is supposed to provide sequenced, duplicate-suppressed, error-free, flow-controlled communication over arbitrarily interconnected communication networks.

A close protocol which is provided by Network streams is also used to implement the communication interface. This method of terminating dialog on a stream is suggested in the NS Internet Protocol Specification. But use of these routines is considered optional.

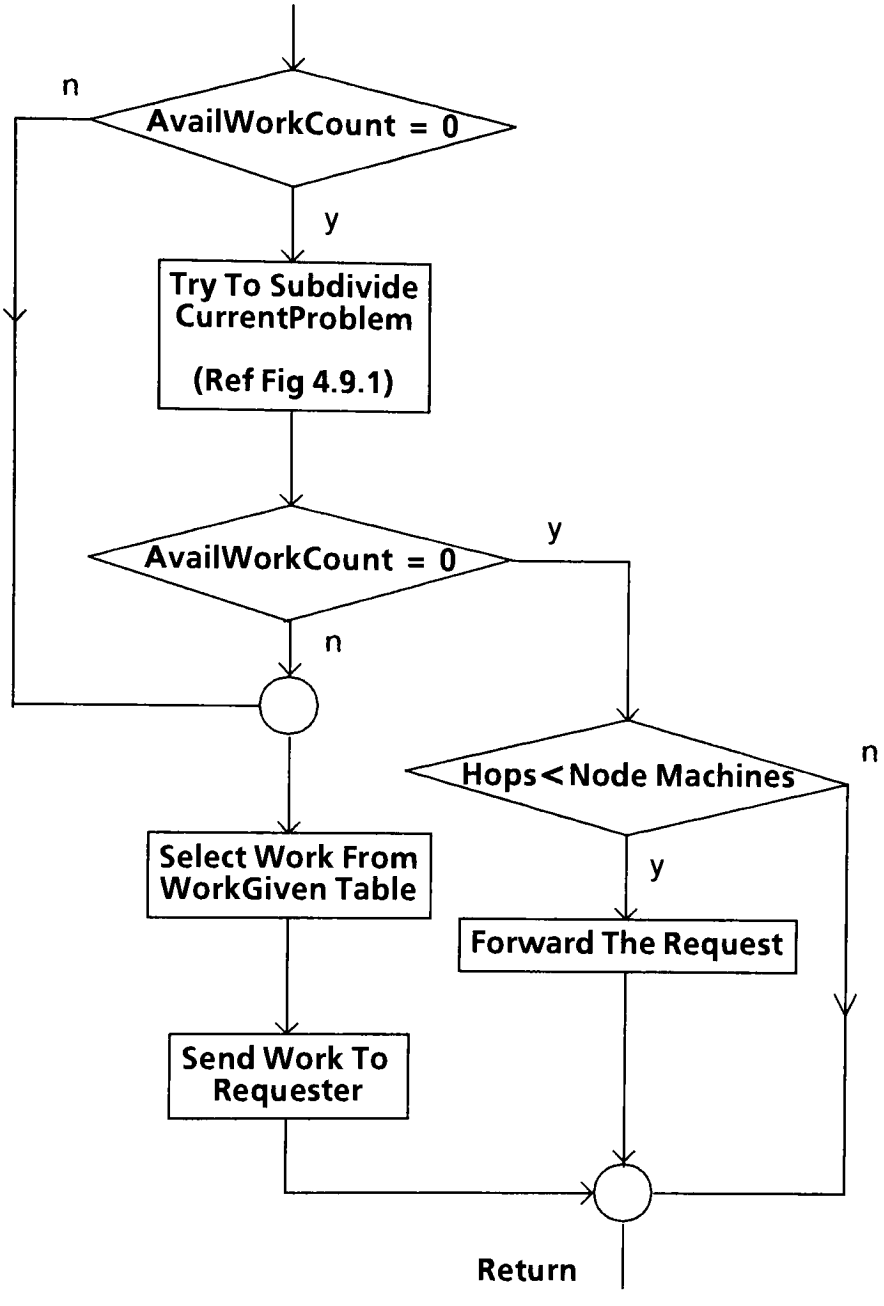
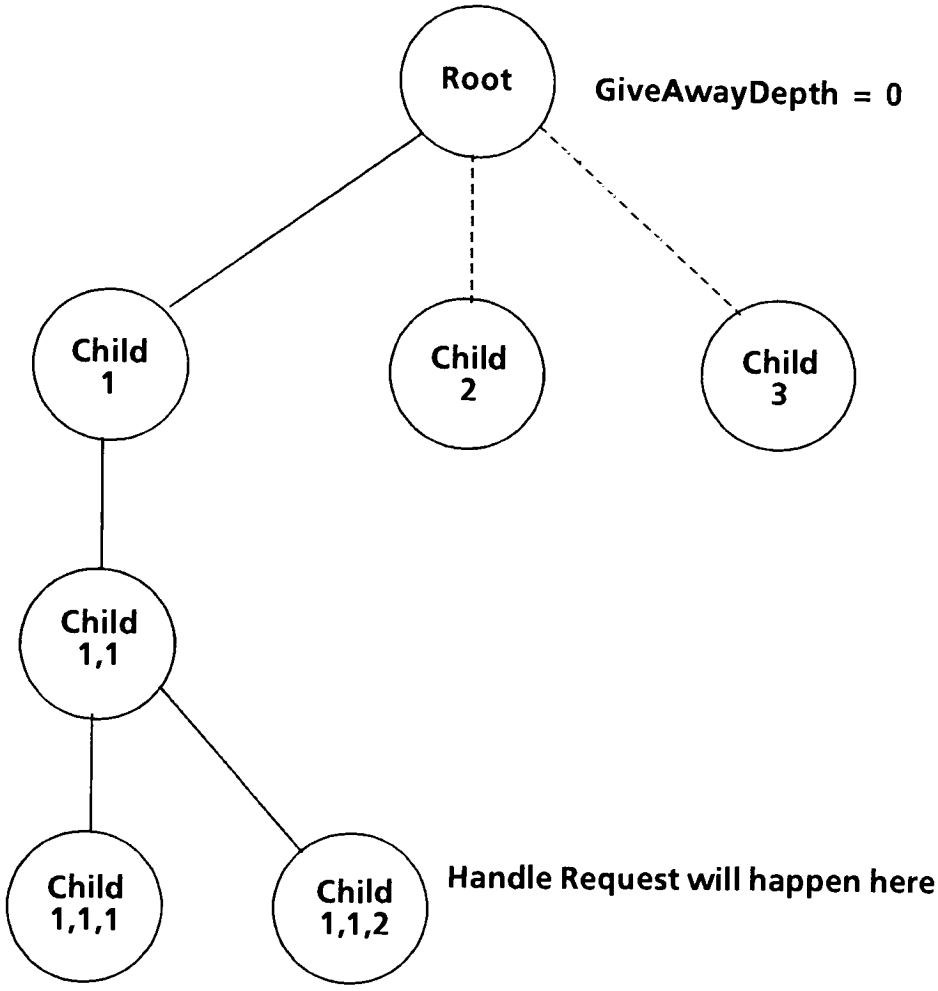



**Figure 4.9 Flow Chart For HandleRequest**




1. Try to subdivide at GiveAwayDepth (Root)
2. If Root is dividable, Child 1 and his siblings (if any) will store in WorkGiven. All Siblings will be marked available.
3. If Root can not be divided, it will increase GiveAwayDepth, try to subdivide at Child1.

## Figure 4.9.1 Subdivide Work In HandleRequest

## 5. USER INTERFACE

To Write an application, the user has to supply the following procedures: ApplicFinish, AcceptRoot, AcceptAnswer, FirstProb, Generate, UseNewInfo, NonTrivial, Combine, Update, ApplicInit, PrintProblem, PrintAnswer. These procedures are similar to what one would need in a sequential program for this application.

The primary procedure is Generate, which DIB calls whenever it needs the next child of a node. DIB indicates whether it is looking for a first child or a later one. The application may respond with a new child or indicate that the node is a leaf, and has no children, or that it has no more children. To start the entire calculation, DIB calls FirstProb to generate the root of the search tree. To print the result of the calculation, DIB calls PrintAnswer on the root node or on the host. The following declarations define DIB's interface to the application program. Appendix B shows two application programs, the eight-queens problem and the traveling-salseman problem.

** Type and Variable

ProblemType: ... the application one wishes to store in a node
InfoType: ... global information format, if any

** The following routines specify initial setup and what to do with results; they are executed on the Host machine (supervisor)

FirstProb: PROCEDURE[Size: INTEGER] RETURNS[P: ProblemType];

Generate the root P of the problem tree, where Size is a parameter selecting which problem of a class is intended. For example, the eight-queens problem can let Size indicate the dimensions of the chessboard.

AcceptAnswer: PROCEDURE[P: ProblemType];

P has information worth recording.

AcceptRoot: PROCEDURE[P: ProblemType];

P contains the result of the entire search.

ApplicFinish: PROCEDURE[];

Used to print final message or do housekeeping.

** DIB calls the following routines on node machines (work)

Generate: PROCEDURE[
First: BOOLEAN,
Parent: LONG POINTER TO ProblemType,
Child: LONG POINTER TO ProblemType]
RETURNS[Done: BOOLEAN];

If First, set Child to first child of Parent. Otherwise, set Child to the next child of Parent. If there are no more children, set Done instead. Generating a child may modify local data within Parent.

UseNewInfo: PROCEDURE[Info: InfoType];

Make use of Info, which was broadcast by a different machine. This information is typically used to improve a bound.

NonTrivial: PROCEDURE[P: ProblemType] RETURNS[BOOLEAN];

Return true if P will take significant computation to complete. DIB will not distribute a problem that the application considers trivial.

Combine: PROCEDURE[
Child: LONG POINTER TO ProblemType,
Parent: LONG POINTER TO ProblemType]
RETURNS[Update: BOOLEAN];

Child has just finished; record any information to be passed up the tree in Parent. If Parent's live children need to be notified asynchronously of Parent's new situation, set Update to true.

Update: PROCEDURE[
Child: LONG POINTER TO ProblemType,
Parent: LONG POINTER TO ProblemType]
RETURNS[Again: BOOLEAN];

Parent has just been modified; Child is already live. Update Child's information based on the Parent. If Child's live children must also be notified of its change, set Again to true.

** The following routines are used for initialization, debugging, and printing results. They are invoked both on the Host and work (Node) Machine.

ApplicInit: PROCEDURE[];
Initialize any data structures.

PrintProblem: PROCEDURE[P: ProblemType];
Print P's problem description.

PrintAnswer: PROCEDURE[P: ProblemType];
Print P's answer values.

** The following routines are provided by DIB on work machines and may be called by the application, for example, during Generate or Combine.

BroadcastInfo: PROCEDURE[Info: InfoType];
Lets the application in all machines know about this new information. This facility is used in branch and bound.

ReportResult: PROCEDURE[P: ProblemType];
Causes the work machine to send P to the Host machine, where AcceptResult will be invoked.

## 6. EXPERIMENTS

Three applications have been implemented and used to test the DIB package. Their execution results are given on Appendix C. The application programs for the eight queens problem and the traveling salesman problem are given on Appendix B. Statistics data for each application is shown in this section and the eight-queens problem is also used as an example to show how to write an application program.

### 6.1 Description Of The Applications

1. The eight queens problem

   Find all possible arrangements of N (chess) queens on an N × N board such that no queen can capture another queen. A count of solutions is passed up the tree.

| Example Result (Queens Problem) N = 4 | | | |
|---|---|---|---|
| | Q2 | Q3 | |
| Q1 | | | Q4 |
| Q1 | | | Q4 |
| | Q2 | Q3 | |

The most important part in an application program is how to generate children at each node. The comments shown in the step (3) describe the algorithm to attack this problem. To write the application program, the user needs to do the following:

(1) Type and variable declarations

```
MaxProbSize: INTEGER = 20; --no more than this many queens
QueenArray: TYPE = ARRAY[1..MaxProbSize] OF INTEGER;
<<store the solution, the number represents the queen's position in
each column>>

ProblemType: TYPE = RECORD[
    LeafCount: LONG INTEGER,
    <<It is a synthesized value generated at the leaves of the tree and
    is passed up as a result of calls to combine. It counts the number of
    arrangements that the queens can be put on the board.>>
    ProbSize: INTEGER, --dimensions of the chess board
    Length: INTEGER, --how many queens have been placed
    Queens: QueenArray]; -- store queen's position

InfoType: TYPE =
            PACKED ARRAY[0..1] OF CHARACTER; --pseudo data
<<InfoType is not needed in queens application, but it is needed by
DIB program.>>

Title: LONG STRING ← "Queens"L;
```

(2) Code the root problem

```
FirstProb: PUBLIC PROCEDURE[Size: INTEGER]
                              RETURNS[P: ProblemType] =

  BEGIN
    P.LeafCount ← 0;
    P.ProbSize ← Size;
    P.Length ← 0;
  END;
```

(3) How to generate children at each node

```
Generate: PUBLIC PROCEDURE[
                        First: BOOLEAN,
                        Parent: LONG POINTER TO ProblemType,
                        Child: LONG POINTER TO ProblemType]
                    RETURNS[Done: BOOLEAN] =

BEGIN
    Seq: INTEGER;
```

```
  ThisQueen, ThatQueen: INTEGER;
  Bad: BOOLEAN;

  --what to do during the search

  IF First AND (Parent↑.Length = Parent↑.ProbSize) THEN

    {-- find a leaf
       Done ← TRUE;
       Parent↑.LeafCount ← 1; -- set leaf count
       ReportResult[Parent↑]}  -- report to Host

  ELSE {--not a leaf

          IF First THEN -- expect to find a child
            {Child↑ ← Parent↑; -- copy parent data to child
             {OPEN Child↑;
              Length ← Length + 1; -- update child's length
              Queens[Length] ← 0}; -- initialize
              First ← FALSE};

            -- search for a reasonable place for current queen

          {OPEN Child↑;
           LeafCount ← 0; -- initialize
           DO
              Queens[Length] ← Queens[Length] + 1;
              ThisQueen ← Queens[Length];
              IF ThisQueen > ProbSize THEN
                  {Done ← TRUE;
                   EXIT};
              Seq ← 1;
              Bad ← FALSE;
              WHILE Seq < Length

                -- check if Queens[Seq] conflicts with Queens[Length]

                DO
                    ThatQueen ← Queens[Seq];
                    IF (ThatQueen = ThisQueen) OR --check row

                       -- check diagonal position
                        (ThatQueen + Seq  Length = ThisQueen) OR
                      (ThatQueen - Seq + Length = ThisQueen) THEN
                    {Bad ← TRUE;
                    EXIT};
                    Seq ← Seq + 1;
               ENDLOOP;
              IF NOT Bad THEN
                {Done ← FALSE;
                 EXIT};
          ENDLOOP}};
END;
```

(4) How to combine the result from children

```
Combine: PUBLIC PROCEDURE[
                    Child: LONG POINTER TO ProblemType,
                    Parent: LONG POINTER TO ProblemType]
                    RETURNS[Update: BOOLEAN] =

  BEGIN
    Parent↑.LeafCount ← Parent↑.LeafCount + Child↑.LeafCount;
    Update ← FALSE;
  END;
```

(5) How to print the answer

```
PrintProblem: PUBLIC PROCEDURE[P: ProblemType] =

    BEGIN
     Seq: INTEGER ← 1;

     {OPEN P;  -- print the queens' position
      WHILE Seq < = Length
        DO
        WriteChar[' ];
        WriteDecimal[Queens[Seq]];
        Seq ← Seq + 1;
        ENDLOOP;
      NewLine[]};
    END;

PrintAnswer: PUBLIC PROCEDURE[P: ProblemType] =

    BEGIN  -- print how many arrangements
        WriteLongDecimal[P.LeafCount];
        WriteString[" Leaves  "L];
    END;
```

(6) Other useful procedures

```
AppliclInit: PUBLIC PROCEDURE[] =
    BEGIN
     WriteString[Title];
     WriteString["    "];
     NewLine[];
    END;

NonTrivial: PUBLIC PROCEDURE[P: ProblemType]
                                RETURNS[BOOLEAN] =
    BEGIN
        RETURN[P.Length < P.ProbSize - 2];
    END;
```

```
AcceptRoot: PUBLIC PROCEDURE[P: ProblemType] =
   BEGIN
      PrintProblem[P];
      PrintAnswer[P];
   END;

AcceptAnswer: PUBLIC PROCEDURE[P: ProblemType] =
    BEGIN
      PrintProblem[P];
    END;
```

(7) Refer to Appendix B, put them in Definition and Implementation module.

2. Knights tour

   Find all possible tours a (chess) knight can take covering the whole board except k squares without repeating a square. (This is basically a Hamiltonian path problem.)

| Example Result (Knights Tour) k = 3 | | | |
|---|---|---|---|
| K10 | K5 | K12 | K3 |
| | K2 | K9 | K6 |
| K8 | K11 | K4 | K13 |
| K1 | | K7 | |

3. The traveling-salesman problem

   Find a minimal-cost tour of cities. Inter-city costs are initialized from a uniform distribution in [0..1]. The application uses a branch and bound algorithm, broadcasts each newly discovered minimum path cost, and generates children of a node in closest-first order. The best solution found is passed up the tree.

| The Costs Between Inter-city | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0.51387 | 0.17574 | 0.30865 | 0.53453 | 0.94763 | 0.17174 | 0.70223 |
| 1 | 0.82295 | 0.15193 | 0.62548 | 0.31468 | 0.34690 | 0.91720 | 0.51976 |
| 2 | 0.41450 | 0.46352 | 0.97916 | 0.12644 | 0.21264 | 0.95845 | 0.73746 |
| 3 | 0.44188 | 0.91502 | 0.57225 | 0.11884 | 0.56977 | 0.25205 | 0.49586 |
| 4 | 0.77343 | 0.24483 | 0.34282 | 0.22999 | 0.29788 | 0.30455 | 0.88721 |
| 5 | 0.15877 | 0.27990 | 0.13532 | 0.86419 | 0.75018 | 0.20800 | 0.13996 |
| 6 | 0.39541 | 0.43220 | 0.12714 | 0.45767 | 0.23783 | 0.98603 | 0.65283 |

| Example Result (Traveling-Salesman)<br>7 Cities Visited |
|---|
| New Path 0 5 2 3 6 4 1 0 (2.23497) |
| New Path 0 5 2 3 4 1 6 0 (2.16327) |
| New Path 0 5 2 4 1 3 6 0 (1.97048) |
| New Path 0 5 6 2 4 1 3 0 (1.65287) |
| Best path 0 5 6 2 4 1 3 (1.65287) |

## 6.2 Results

Table 6.1 collects the running times for the queens problem (queen's number is changed from 8 to 12) for 1 to 5 workstations. The times were measured by the Host machine. From the table, we can know that if the problem size is small (if queen's number is less than 10), it is not worth to use more than one machine to run the queens problem. Because the time is needed for synchronization, subdividing problem, and communication

between machines, the running time increases with the number of machines for small problem size.

Figure 6.1 shows timings (in seconds) for the twelve-queens problem. The time curve is measured in seconds. The figure also shows the efficiency, which is defined as

**{time for 1 machine} over {n × (time for n machines)}**

The efficiency is generally above 90%, and seems to degrade quite slowly as extra machines are added.

Table 6.2 and Figure 6.2 show the statistics data gathered from the knights tours problem and Table 6.3 and Figure 6.3 show the statistics data gathered from the traveling salesman problem.

| Queen # | Running Times for 1 to 5 workstations | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| 8 | 0:2.39 | 0:5.05 | 0:5.04 | 0:6.27 | 0:7.41 |
| 9 | 0:12.89 | 0:9.95 | 0:10.26 | 0:11.18 | 0:10.27 |
| 10 | 1:3.7 | 0:40.5 | 0:31.4 | 0:23.9 | 0:21.5 |
| 11 | 5:45.6 | 2:57.5 | 2:3.4 | 1:35.2 | 1:23.0 |
| 12 | 33:51.8 | 16:59.9 | 11:27.5 | 8:36.5 | 7:0.0 |

Table 6.1 Queens Problem Testing Result

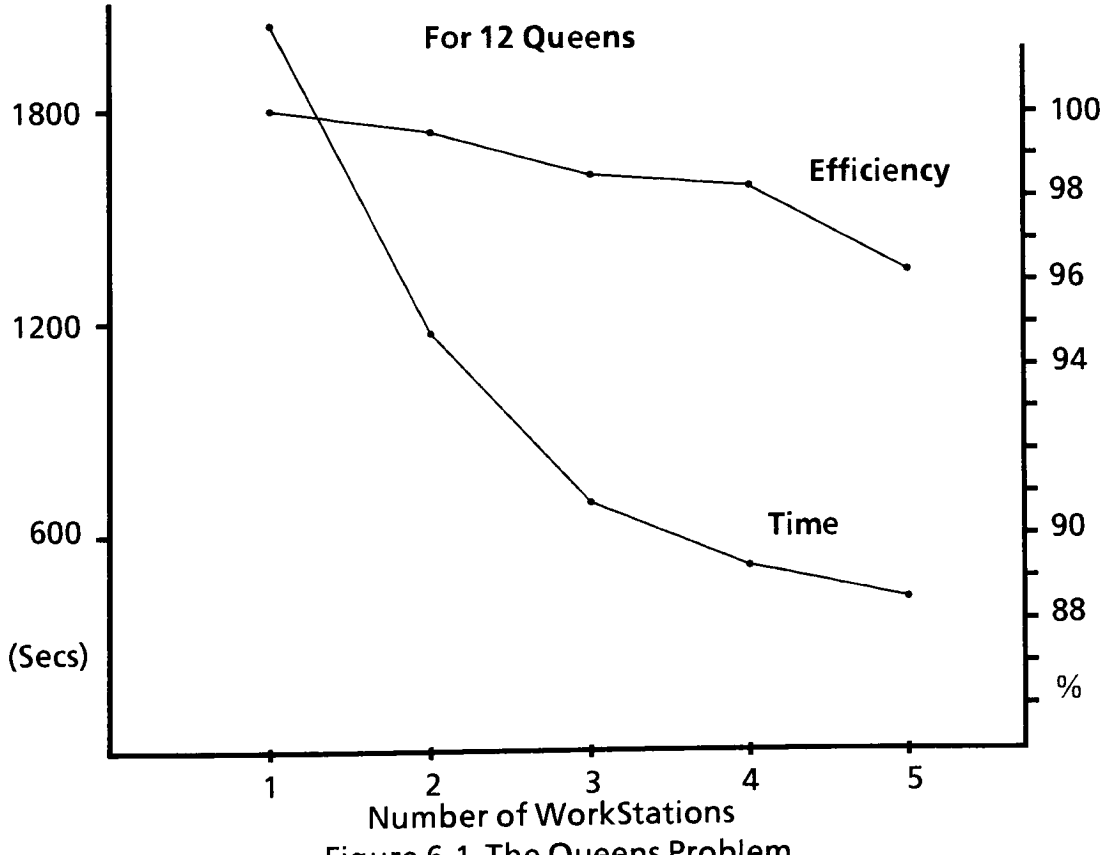



Figure 6.1 The Queens Problem

| Board Dimension | Running Times for 1 to 5 workstations | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| 3 | 0:0:0.534 | 0:0:2.15 | 0:0:2.98 | 0:0:4.927 | 0:0:4.335 |
| 4 | 0:0:6.034 | 0:0:5.471 | 0:0:8.036 | 0:0:9.016 | 0:0:11.065 |
| 5 | 1:11:1.0 | 0:35:38.9 | 0:24:30.3 | 0:18:28.5 | 0:15:18.2 |

Table 6.2 Knights Tour Testing Result

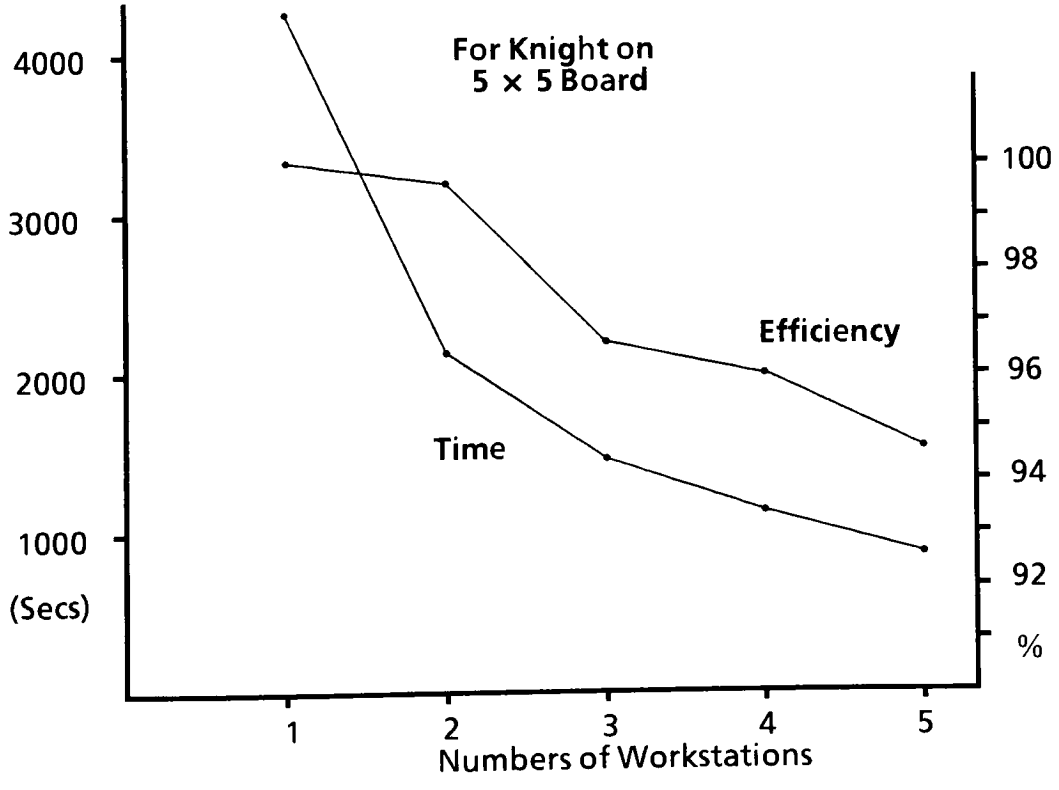



Figure 6.2 Knights Tour Problem

| Cities Visited | Running Times for 1 to 4 workstations | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| 9 | 0:0:38.18 | 0:0:21.91 | 0:0:20.78 | 0:0:15.74 | |
| 10 | 0:2:43.37 | 0:1:30.64 | 0:1:27.21 | 0:0:53.67 | Machines |
| 11 | 0:14:13.91 | 0:7:16.25 | 0:5:0.18 | 0:3:57.76 | Are |
| 12 | 0:19:51.24 | 0:10:7.55 | 0:7:4.57 | 0:5:35.63 | Not |
| 13 | 0:44:21.37 | 0:22:27.95 | 0:15:54.98 | 0:11:53.13 | Available |
| 14 | 2:14:1.0 | 1:9:19.0 | 0:48:2.464 | 0:37:9.33 | |

Table 6.3 Traveling Salesman Testing Result

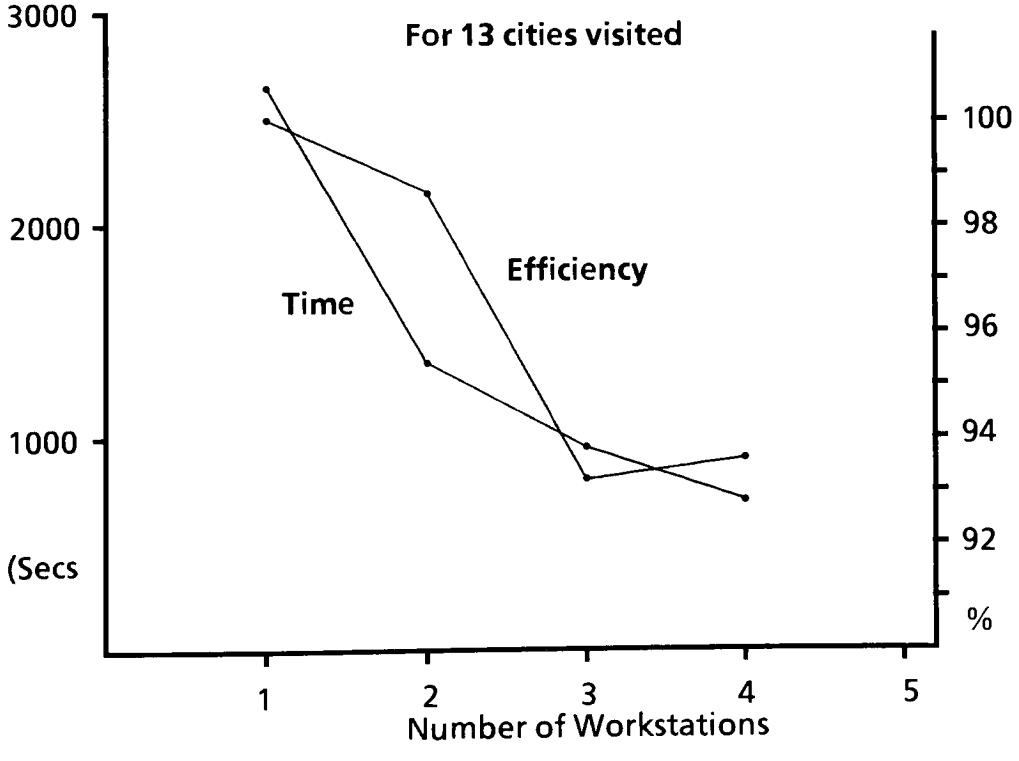



Figure 6.3 Traveling Salesman Problem

During the execution, each Node workstation measures elapsed time in several categories: time spent in communication (generating messages, Processing requests), time spent idly waiting for another work machine to grant a request for work, time spent idly before discovering termination, and time spent computing. The following tables present the computing and communication timings for 5 Node workstations jointly solving a 12-queen problem and for 4 Node workstations jointly solving a 14-city sorted traveling salesman problem. The tables show how equally work is distributed among the workstations and only a small fraction of the time is spent in communication

| Node | Communication time | Computation time |
|---|---|---|
| 1 | 0:9.887 | 6:53.212 |
| 2 | 0:10.306 | 6:48.917 |
| 3 | 0:7.557 | 6:50.653 |
| 4 | 0:16.595 | 6:49.524 |
| 5 | 0:8.845 | 6:50.334 |

Table 6.4 Distribution of work for 12 queens

| Node | Communication time | Computation time |
|---|---|---|
| 1 | 0:5.878 | 36:55.63 |
| 2 | 0:9.736 | 36:45.882 |
| 3 | 0:9.157 | 36:45.373 |
| 4 | 0:11.812 | 36:40.635 |

Table 6.5 Distribution of work for 14-city visited

# 7. IMPLEMENTATION EXPERIENCES

There are several issues that are worth mentioning in using the Mesa programming language to implement this project. They are described as follows.

## 7.1 The Size Of Variables

The maximum size of a variable is around 4000 words; the user may declare a variable as " test: ARRAY[1..4000] OF INTEGER " and will not get any error message from the compiler. If the variable size does exceed this limit, the compiler will print out "variable or field too big" message to the user.

The user may use the Heap which is the primary storage allocator in Mesa to allocate and free blocks of storage of arbitrary size. For example, the user can do the follow to let the variable size exceed 4000 words.

```
testType: TYPE = ARRAY [1..8000] OF INTEGER; -- declaration

test: LONG POINTER TO testType; -- declaration

test ← zone.NEW[testType];     -- in program

zone.Free[@test]; -- in program
```

## 7.2 How To Deal With Real Numbers

Users may want to use real numbers in their application programs. The real number operations and functions such as FAdd, FSub, FMul, FDiv, FRem, FComp, Exp, Log, Ln, SqRt, Root and Power can be found in the Real.mesa and RealFns.mesa under the Fiat/**XDE/ComSoft/Public** directory.

The procedure "InitReals" must be called before any floating point operations are called. It is ok to call InitReals more than once.

## 7.3 How To Use Variant Records

Users often find it convenient to combine information of different types. The user can use the following steps to declare variant RECORD types, to assign values to variant RECORDs, and to access the fields of variant RECORDs.

1. Declaring variant RECORDs

The declaration for Message is a variant RECORD:

```
Message: TYPE = RECORD[                              -- declaration
    specifics: SELECT type: MessageClass FROM    -- no common part
        ProblemMessage = > [
            PartSize: INTEGER,
            DebugLevel: INTEGER],
        InterNodeMessage = > [
            Asker: INTEGER,
            Redundancy: INTEGER],
    ENDCASE];
```

2. Allocation of variant RECORDs

```
MsgBuffer: RECORD[
        source: INTEGER,
        dest: INTEGER,
        message: Message];
```

3. Assignment to variant RECORD variables

```
a: ProblemMessage Message;          -- declaration
                                    -- It now is a bound variant
a.PartSize ← 5;
a.DebugLevel ← 2;
```

```
MsgBuffer.message ← a;              -- you can do it in this way
```

4. Accessing the fields of a variant RECORD variable

```
WITH a: MsgBuffer.message SELECT FROM
   ProblemMessage = > {
         x ← a.PartSize;            -- process the data
         y ← a.DebugLevel};
   << OR InterNodeMessage = > {
         *** }; >>
   ENDCASE;
```

## 7.4 About Process.Yield

The Mesa process mechanism does not attempt to allocate processor time fairly among processes of equal priority. A process itself will yield the processor to other processes of equal priority whenever it faults, Pauses or WAITs. In the Node program, the Receiver and the main program run concurrently. When the problem size is large, as in queen sizes equal 12 or larger, other Nodes could not connect to Node 1 until Node 1 nearly finished all the computation. It looks like the Receiver has the problem getting to the processor.

Process.Yield places the calling process at the rear of the queue of ready-to-run processes of the same priority. Thus, all other ready processes of the same priority will run before the calling process next runs. After using it in the main program to force the scheduling, it did solve the connection problem but it increased overhead by 5 to 10 percent.

## 8. FURTHER RESEARCH AND CONCLUSIONS

There are several directions for advancing this software package.

1. Adapting DIB to similar application areas. These applications may be an alpha-beta search problem, or a Petri-net reachability problem.

2. Distributing work not solely by request from underloaded nodes, but also by initiative of overloaded nodes. This facility will help shed load from "hot spots", which occur near the end of computation and when machines have failed.

3. Adapting DIB to be used as an instructional tool in networking and distributed computing courses.

DIB is an attempt at providing an easily used facility for writing distributed programs involving backtracking. The distributed part of DIB is completely hidden from the user, making it especially suitable for programmers without expertise in distributed or parallel programming. The distribution of work in DIB is dynamic, making it suitable for any environments. DIB requires minimal support from the distributed operating system and it is relatively portable.

The experimental results indicate that for simple programs DIB is an efficient and easily used tool for automatically distributing work. Exhaustive search is distributed uniformly with very little performance penalty. Application that uses branch and bound should generate more worthy children first to take advantage of DIB's search order. The feature of fault tolerance makes DIB very robust.

## BIBLIOGRAPHY


1. R. A. Finkel and U. Manber, DIB A Distributed Implementation of backtracking, Proceedings of the 5th International Conference on Distributed Computing Systems, 1985

2. G. M. Baudet. The Design and Analysis of Algorithms for Asynchronous Multiprocessors. Department of Computer Science. Carnegie-Mellon University (April 1978)

3. R. A. Finkel and J. P. Fishburn. "Parallelism in Alpha-Beta Search." Journal of Artificial Intelligence 19(1) (September 1982)

4. S. G. Akl, D. T. Barnard and R. J. Doran. "Simulation and analysis in deriving time and storage requirements for a parallel alpha-beta algorithm." Proc. 1980 international Conference on Parallel Processing. (August 1980)

5. P Moller-Nielsen and J. Staunstrup. "Experiments with a Multiprocessor." Technical Report PB-185, Computer Science Department. AArhus University (November 1984)

6. G. B. Adams III, H. J. Siegel, M. Imai, I, Fukumura, and Y. Yoshida, "A Parallelized branch-and-bound algorithm: implementation and efficiency." Systems, Computers, Controls 10(3) pp. 270-2277 (1979)

7. E. L. Lawer and D. Wood, "Branch and Bound methods: a survey", Operations Research, 1966

8. J. A. Fishburn and R. A. Finkel, "Quotient Networks", IEEE Transactions on Computers, April 1981

9. N. Wirth, Modula: a language for modular multiprogramming, Software Practice and Experience, 1977

10. N. Wirth, Design and Implementation of Modula, Software Practice and Experience, 1977

11. Mesa Language Manual, Xerox Corporation, 1984

12. Mesa Programmer's Manual, Xerox Corporation, 1985

13. Pilot Programmer's Manual, Xerox Corporation, 1985

14. Raphael Finkel. Robert Cook. David DeWitt. Nacy Hall. and Lawrence Landweber, "Wisconsin Modula: Part III of the First Report on the crystal project." Technical Report 501. University of Wisconsin-Madison Computer Sciences (April 1983).

## Appendix A: USER MANUAL

There are certain steps to be followed by the user of this package.

Step 1: Write application program

1. Application program should use "ApplicDfs.mesa" as the name of Definition module, and use "ApplicImpl.mesa" for Implementation module.
2. Follow the description of each procedure in "USER INTERFACE", and use the same procedure name to write each procedure.
3. Refer to the application program examples in Appendix B to get more information.

Step 2: Compile the application program and Bind it with DIB package: retrieve DIB from "Audi/Projects/DIB/Programs" directory

DIB package contains the following Definition modules

1. IO.mesa
2. TimeMeasureDfs.mesa
3. ApplicDfs.mesa (Supplied by user)
4. DeweyDfs.mesa
5. HostDfs.mesa
6. InterfaceDfs.mesa
7. HInterfaceDfs.mesa
8. NodeDfs.mesa
9. NInterfaceDfs.mesa

DIB package contains the following Implementation modules

1. HInterfaceImpl.mesa
2. IOImpl.mesa
3. DeweyImpl.mesa
4. ApplicImpl.mesa (Supplied by user)
5. TimeMeasureImpl.mesa

6. HostImpl.mesa
7. NInterfaceImpl.mesa
8. NIOImpl.mesa
9. NodeImpl.mesa

DIB package contains two configuration modules

1. DIBHost.config 2. DIBNode.config

DIB package needs the following tool modules: find and retrieve them from the "Fiat/XDE/5.0/..." directory.

1. String.bcd
2. Stream.bcd
3. NetworkStream.bcd
4. Heap.bcd
5. Tool.bcd
6. TTYSW.bcd
7. Process.bcd
8. TTY.bcd
9. System.bcd
10. Put.bcd
11. Exec.bcd
12. FormSW.bcd
13. AddressTranslation.bcd
14. Format.bcd
15. Time.bcd
16. RandomDefs.bcd
17. RandomProg.bcd
18. Real.bcd
19. RealImpl.bcd
20. Ascii.bcd
21. Window.bcd
22. Environment.bcd
23. UserTerminal.bcd
24. ToolWindow.bcd

User should compile and bind these modules in the following steps

1. Active "CommandCentral" window if it is in tiny or inactive state. Otherwise, get it from "Executive" window.
2. Enter those Definition modules in the following order to the "compile:" field on the CommandCentral window; IO TimeMeasureDfs ApplicDfs DeweyDfs HostDfs InterfaceDfs HInterfaceDfs NodeDfs NInterfaceDfs, and invoke "Compile!" command.
3. Enter those Implementation modules to the "Compile:" field, the order is not important, and invoke "Compile!" command.
4. Enter DIBHost and DIBNode to the "Bind:" field, and invoke "Bind!" command. Ignore if the Binder gives the message "BroadcastInfo or ReportResult" are not bindable when Binder tried to bind DIBHost.config.
5. If everything is OK, the Binder will generate two executable programs, DIBHost.bcd and DIBNode.bcd.
6. If there are compiling errors from ApplicDfs.mesa, then redo steps 2-4.
7. If there are compiling errors from ApplicImpl.mesa, then only ApplicImpl.mesa needs to be recompile and redo step 4.

Step 3: Run application program

1. Load DIBHost.bcd or DIBNode.bcd (via FileTool) onto those workstations which are going to be used.
2. Enter DIBHost to the "Run:" field of CommandCentral window, and invoke "Run!" command at only one workstation.
3. Enter DIBNode to the "Run:" field of CommandCentral window, and invoke "Run!" command at other workstations.
4. Set Node (work) workstations
   Enter the workstation's name shown on the monitor upper left corner to the "I am:" field and invoke "Assert!" command.

5. Set Host (supervisor) workstation

   (1) Enter the workstation's name shown on the monitor upper left corner to the "I am:" field and invoke "Assert!" command.

   (2) Follow the instructions on the monitor screen to input data or to make a selection. For example,

   a. "Please input Node Number"
   Enter the number of node (work) workstations (exclude the host).

   b. "Please input the name of each node - First is Host"
   Enter the Host name first, then enter each node workstation's name.

   c. Selection
   c: count, only the final result will be reported to Host.
   f: full, each possible answer will be reported to Host.
   d: debug level; trace the execution of the application program. The possible value is 1,2,3, and 6. The bigger the number is, the more details of the execution will be shown on the screen.
   h: helpers, number of helpers when out of work. Due to the number of workstations that can be used, helpers is set to 1 by default.
   n: new problem, run the application program with different problem sizes.
   q: quit

Step 4: Back to Copilot

1. From Host, enter 'q' then deactivate the window and press the SHIFT-STOP keys.
2. From Node, deactivate the window and press the SHIFT-STOP keys.

**NOTE:**

1. DIB can only use these workstations: ALVIS, LIT, GALAR, FJALAR, ANDVARI, EITRI, BROKK, MODSOGNIR, DURIN.
2. These workstations use fixed network addresses to communicate with each other. These addresses are set in the HInterfaceImpl.mesa and NInterfaceImpl.mesa programs.
   If workstations are moved to another place or are connected to other sockets, users need to find the new network addresses and to revise them in the HInterfaceImpl.mesa and NInterfaceImpl.mesa programs.
3. If the "Sender ConnectionFail" message occurs during the "Set up environment" step, the user may need to check the network addresses.

## Appendix B: Sample Application Programs

### B.1 Application Example 1 (Eight-Queens Problem)

Part 1 Definition Part -- File name: ApplicDfs.mesa

```
ApplicDfs: DEFINITIONS =

  BEGIN
    MaxProbSize: INTEGER = 20;
    QueenArray: TYPE = ARRAY[1..MaxProbSize] OF INTEGER;
    ProblemType: TYPE = RECORD[
        LeafCount: LONG INTEGER,
        ProbSize: INTEGER,
        Length: INTEGER,
        Queens: QueenArray];
    InfoType: TYPE = PACKED ARRAY[0..1] OF CHARACTER; -- pseudo data

    ApplicFinish: PROCEDURE[];

    AcceptRoot: PROCEDURE[P: ProblemType];

    AcceptAnswer: PROCEDURE[P: ProblemType];

    FirstProb: PROCEDURE[Size: INTEGER] RETURNS[P: ProblemType];

    Generate: PROCEDURE[First: BOOLEAN,
                        Parent: LONG POINTER TO ProblemType,
                        Child: LONG POINTER TO ProblemType]
                       RETURNS[Done: BOOLEAN];

    UseNewInfo: PROCEDURE[Info: InfoType];

    NonTrivial: PROCEDURE[P: ProblemType] RETURNS[BOOLEAN];

    Combine: PROCEDURE[Child: LONG POINTER TO ProblemType,
                       Parent: LONG POINTER TO ProblemType]
                      RETURNS[Update: BOOLEAN];

    Update: PROCEDURE[Child: LONG POINTER TO ProblemType,
                      Parent: LONG POINTER TO ProblemType]
                     RETURNS[Again: BOOLEAN];

    ApplicInit: PROCEDURE[];

    PrintProblem: PROCEDURE[P: ProblemType];

    PrintAnswer: PROCEDURE[P: ProblemType];
```

```
END.
```

Part 2 Implementation Part -- File name: ApplicImpl.mesa

```
DIRECTORY
 ApplicDfs,
 NodeDfs,
 IO;

ApplicImpl: PROGRAM
 IMPORTS NodeDfs, IO
 EXPORTS ApplicDfs =

 BEGIN
  OPEN NodeDfs, ApplicDfs, IO;
  Title: LONG STRING ← "Queens"L;

  PrintProblem: PUBLIC PROCEDURE[P: ProblemType] =
   BEGIN
    Seq: INTEGER ← 1;

    {OPEN P;
     WHILE Seq < = Length
       DO
       WriteChar[' ];
       WriteDecimal[Queens[Seq]];
       Seq ← Seq + 1;
       ENDLOOP;
     NewLine[]};
   END;

  PrintAnswer: PUBLIC PROCEDURE[P: ProblemType] =
   BEGIN
    WriteLongDecimal[P.LeafCount];
    WriteString[" Leaves "L];
   END;

  Generate: PUBLIC PROCEDURE[First: BOOLEAN,
                             Parent: LONG POINTER TO ProblemType,
                             Child: LONG POINTER TO ProblemType]
                             RETURNS[Done: BOOLEAN] =
   BEGIN
    Seq: INTEGER;
    ThisQueen, ThatQueen: INTEGER;
    Bad: BOOLEAN;

    IF First AND (Parent ↑ .Length = Parent ↑ .ProbSize) THEN
      {Done ← TRUE;
       Parent ↑ .LeafCount ← 1;
       ReportResult[Parent ↑ ]}
    ELSE { --not a leaf
```

```
      IF First THEN
        {Child↑ ← Parent↑;
         {OPEN Child↑;
          Length ← Length + 1;
          Queens[Length] ← 0};
         First ← FALSE};
        {OPEN Child↑;
         LeafCount ← 0;
         DO
           Queens[Length] ← Queens[Length] + 1;
           ThisQueen ← Queens[Length];
           IF ThisQueen > ProbSize THEN
             {Done ← TRUE;
              EXIT};
           Seq ← 1;
           Bad ← FALSE;
           WHILE Seq < Length
             DO
             ThatQueen ← Queens[Seq];
             IF (ThatQueen = ThisQueen) OR
                    (ThatQueen + Seq  Length = ThisQueen) OR
                    (ThatQueen - Seq + Length = ThisQueen) THEN
               {Bad ← TRUE;
                EXIT};
             Seq ← Seq + 1;
             ENDLOOP;
           IF NOT Bad THEN
             {Done ← FALSE;
              EXIT};
         ENDLOOP}};
  END;

FirstProb: PUBLIC PROCEDURE[Size: INTEGER] RETURNS[P: ProblemType] =
  BEGIN
   P.LeafCount ← 0;
   P.ProbSize ← Size;
   P.Length ← 0;
  END;

Combine: PUBLIC PROCEDURE[Child: LONG POINTER TO ProblemType,
                          Parent: LONG POINTER TO ProblemType]
                         RETURNS[Update: BOOLEAN] =

  BEGIN
   Parent↑.LeafCount ← Parent↑.LeafCount + Child↑.LeafCount;
   Update ← FALSE;
  END;


NonTrivial: PUBLIC PROCEDURE[P: ProblemType] RETURNS[BOOLEAN] =
  BEGIN
   RETURN[P.Length < P.ProbSize - 2];
  END;
```

```
 ApplicInit: PUBLIC PROCEDURE[] =
  BEGIN
   WriteString[Title];
   WriteString[" - "];
   NewLine[];
  END;

 ApplicFinish: PUBLIC PROCEDURE[] =
  BEGIN
  END;

 UseNewInfo: PUBLIC PROCEDURE[P: InfoType] =
  BEGIN
  END;

 AcceptRoot: PUBLIC PROCEDURE[P: ProblemType] =
  BEGIN
   PrintProblem[P];
   PrintAnswer[P];
  END;

 AcceptAnswer: PUBLIC PROCEDURE[P: ProblemType] =
  BEGIN
   PrintProblem[P];
  END;

 Update: PUBLIC PROCEDURE[Child: LONG POINTER TO ProblemType,
                          Parent: LONG POINTER TO ProblemType]
                         RETURNS[Again: BOOLEAN] =

  BEGIN
   Again ← FALSE;
  END;

END.
```

B.2 Application Example 2 (Traveling Salesman Problem)

Part 1 Definition Part -- File name: ApplicDfs.mesa

```
ApplicDfs: DEFINITIONS =

 BEGIN
  MaxProbSize: INTEGER = 20;
  Infinity: REAL = 1.0e30;
  PermArray: TYPE = ARRAY[0..MaxProbSize   1] OF INTEGER;
  ProblemType: TYPE = RECORD[
   ProbSize: INTEGER,
   Length: INTEGER,
   Perm: PermArray,
   Children: PermArray,
   Cost: REAL,
   LeafCount: LONG INTEGER,
   BestCost: REAL,
   BestPerm: PermArray];
  InfoType: TYPE = RECORD[Estimate: REAL];

  ApplicFinish: PROCEDURE[];

  AcceptRoot: PROCEDURE[P: ProblemType];

  AcceptAnswer: PROCEDURE[P: ProblemType];

  FirstProb: PROCEDURE[Size: INTEGER] RETURNS[P: ProblemType];

  Generate: PROCEDURE[First: BOOLEAN,
                      Parent: LONG POINTER TO ProblemType,
                      Child: LONG POINTER TO ProblemType]
                       RETURNS[Done: BOOLEAN];

  UseNewInfo: PROCEDURE[Info: InfoType];

  NonTrivial: PROCEDURE[P: ProblemType] RETURNS[BOOLEAN];

  Combine: PROCEDURE[Child: LONG POINTER TO ProblemType,
                     Parent: LONG POINTER TO ProblemType]
                    RETURNS[Update: BOOLEAN];

  Update: PROCEDURE[Child: LONG POINTER TO ProblemType,
                    Parent: LONG POINTER TO ProblemType]
                   RETURNS[Again: BOOLEAN];

  ApplicInit: PROCEDURE[];

  PrintProblem: PROCEDURE[P: ProblemType];

  PrintAnswer: PROCEDURE[P: ProblemType];

END.
```

Part 2 Implementation Part -- File name: ApplicImpl.mesa

```
DIRECTORY
 ApplicDfs,
 NodeDfs,
 IO,
 Real;

ApplicImpl: PROGRAM
 IMPORTS NodeDfs,IO,Real
 EXPORTS ApplicDfs =

 BEGIN
  OPEN NodeDfs,ApplicDfs,IO,Real;
  InitialDataType: TYPE = ARRAY[0..MaxProbSize  1] OF
       ARRAY[0..MaxProbSize - 1] OF REAL;


  BestInfo: InfoType ← [Infinity];
  Title: LONG STRING ← "Sorted Travel"L;
  InitialData: InitialDataType ←

[[0.51387,0.17574,0.30865,0.53453,0.94763,0.17174,0.70223,0.22643,0.49477,

0.12472,0.08390,0.38965,0.27723,0.36807,0.98344,0.53540,0.76568,0.64647,
    0.76714,0.78024],

[0.82295,0.15193,0.62548,0.31468,0.34690,0.91720,0.51976,0.40115,0.60676,

0.78540,0.93152,0.86992,0.86652,0.67452,0.75840,0.58189,0.38925,0.35563,
    0.20023,0.82693],

[0.41450,0.46352,0.97916,0.12644,0.21264,0.95845,0.73746,0.40906,0.78011,

0.75790,0.95684,0.02810,0.31873,0.75693,0.24299,0.58954,0.04342,0.95602,
    0.31913,0.05936],

[0.44188,0.91502,0.57225,0.11884,0.56977,0.25205,0.49586,0.23673,0.47696,

0.40609,0.87300,0.42696,0.35822,0.38199,0.04318,0.16059,0.52235,0.69658,
    0.09710,0.40085],

[0.77343,0.24483,0.34282,0.22999,0.29788,0.30455,0.88721,0.03667,0.65115,

0.39861,0.67630,0.73258,0.93780,0.23328,0.83848,0.96721,0.77864,0.43152,
    0.67410,0.80936],

[0.15877,0.27990,0.13532,0.86419,0.75018,0.20800,0.13996,0.29459,0.80281,

0.21893,0.56308,0.71560,0.19754,0.98982,0.25004,0.43061,0.75527,0.86093,
    0.89478,0.97809],

[0.39541,0.43220,0.12714,0.45767,0.23783,0.98603,0.65283,0.60425,0.24191,
```

0.45487,0.78996,0.07882,0.47641,0.15259,0.24575,0.94499,0.61402,0.98819,
0.47728,0.79968],

[0.74418,0.38074,0.47989,0.52691,0.09810,0.59421,0.34718,0.14377,0.77954,

0.71100,0.44614,0.70457,0.09531,0.96283,0.55130,0.74026,0.57904,0.63788,
0.78166,0.18790],

[0.30210,0.28281,0.68401,0.29292,0.56539,0.41845,0.30658,0.44453,0.56569,

0.48793,0.60663,0.41586,0.13042,0.25596,0.03576,0.97710,0.11451,0.37805,
0.64671,0.35045],

[0.55305,0.35841,0.56545,0.47563,0.16368,0.61522,0.17217,0.55471,0.29223,

0.87216,0.83506,0.84489,0.89551,0.59476,0.54057,0.16821,0.65496,0.69052,
0.26385,0.10669],

[0.81491,0.19136,0.42330,0.35186,0.83921,0.13733,0.26267,0.17725,0.47992,

0.38017,0.50482,0.50278,0.35192,0.52558,0.12063,0.51956,0.60712,0.73291,
0.55688,0.34413],

[0.80197,0.59099,0.26691,0.67068,0.55215,0.78893,0.88772,0.89000,0.06811,

0.80058,0.90737,0.64413,0.16515,0.30136,0.16628,0.28517,0.84198,0.53632,
0.03635,0.20721],

[0.02124,0.35813,0.62147,0.52003,0.54604,0.15368,0.82336,0.03335,0.02597,

0.37813,0.61633,0.02039,0.62656,0.91520,0.37480,0.72946,0.39581,0.98227,
0.59729,0.11233],

[0.22159,0.79918,0.87066,0.73823,0.01363,0.73956,0.41835,0.36203,0.20391,

0.18316,0.07629,0.11556,0.15912,0.78826,0.04036,0.79064,0.59901,0.40260,
0.22905,0.18280],

[0.61432,0.33190,0.60515,0.96411,0.37807,0.18441,0.30009,0.05420,0.14402,

0.01045,0.88485,0.95802,0.62590,0.95560,0.63104,0.03915,0.35131,0.14634,
0.10602,0.19744],

[0.08395,0.02683,0.94572,0.91962,0.90798,0.86561,0.14893,0.17179,0.06822,

0.65102,0.73687,0.10242,0.16001,0.09397,0.12173,0.02464,0.76231,0.95696,
0.02790,0.64660],

[0.10806,0.42793,0.30973,0.01860,0.88529,0.75787,0.50954,0.16579,0.76291,

0.88083,0.49955,0.87513,0.73477,0.23503,0.05162,0.60548,0.87595,0.50401,
0.67839,0.98939],

[0.60479,0.49634,0.58975,0.89548,0.04463,0.88285,0.10822,0.52011,0.57883,

```
0.00994,0.38707,0.47714,0.19295,0.50789,0.77501,0.35440,0.69776,0.91279,
    0.67096,0.70578],

[0.42689,0.02091,0.21296,0.94759,0.50282,0.19426,0.64472,0.12799,0.26495,

0.33603,0.70369,0.03814,0.95359,0.75474,0.87435,0.63400,0.24353,0.63580,
    0.85049,0.23738],

[0.72095,0.33922,0.05023,0.48501,0.89747,0.24299,0.52751,0.49445,0.85505,

0.34589,0.12384,0.21552,0.11548,0.36322,0.20400,0.43607,0.82828,0.50973,
    0.81951,0.41071]];

  PrintChildren: PROCEDURE[P: ProblemType] =
   BEGIN
    Seq: INTEGER ← 0;

    {OPEN P;
     WriteChar['{];
     WHILE Seq < ProbSize - Length
       DO
       WriteDecimal[Children[Seq]];
       WriteString[" "L];
       Seq ← Seq + 1;
       ENDLOOP;
     WriteString["} "L]};
   END;

  PrintProblem: PUBLIC PROCEDURE[P: ProblemType] =
   BEGIN
    Seq: INTEGER ← 0;

    {OPEN P;
     WHILE Seq < Length
       DO
       WriteChar[' ];
       WriteDecimal[Perm[Seq]];
       Seq ← Seq + 1;
       ENDLOOP;
     WriteString[" ("L];
     WriteFloat[Cost];
     WriteString[") "L];
     IF LeafCount > 0 THEN
      {WriteString[", "L];
       WriteLongDecimal[LeafCount];
       WriteString[" Leaves "L]}};
    PrintChildren[P];
   END;

  SortChildren: PROCEDURE[Ptr: LONG POINTER TO ProblemType] =
   BEGIN
    PrevNode: INTEGER;
    ThisChild: INTEGER;
```

```
 ThisDistance: REAL;
Seq, Seq1: INTEGER;

{OPEN Ptr↑;
 PrevNode ← Perm[Length  1];
 Seq ← 1;
 WHILE Seq < = ProbSize - Length - 1
  DO
  ThisChild ← Children[Seq];
  ThisDistance ← InitialData[PrevNode][ThisChild];
  Seq1 ← Seq - 1;
  WHILE (Seq1 > -1) AND
    (FComp[ThisDistance,InitialData[PrevNode][Children[Seq1]]] = -1)
   DO
   Children[Seq1 + 1] ← Children[Seq1];
   Seq1 ← Seq1 - 1;
   ENDLOOP;
  Children[Seq1 + 1] ← ThisChild;
  Seq ← Seq + 1;
 ENDLOOP};
END;

Generate: PUBLIC PROCEDURE[First: BOOLEAN,
                           Parent: LONG POINTER TO ProblemType,
                            Child: LONG POINTER TO ProblemType]
                           RETURNS[Done: BOOLEAN] =
BEGIN
 Seq: INTEGER;
OldNext: INTEGER;

IF Parent↑.Length = Parent↑.ProbSize THEN
  {OPEN Parent↑;
   Length ← Length + 1;
   Perm[Length - 1] ← 0;
   Cost ← FAdd[Cost,InitialData[Perm[Length-2]][Perm[Length-1]]];
   IF FComp[Cost,BestInfo.Estimate] = -1 THEN
     {BestInfo.Estimate ← Cost;
      BestCost ← Cost;
      BestPerm ← Perm;
      ReportResult[Parent↑];
     BroadcastInfo[BestInfo]}
   ELSE BestCost ← Infinity;
   Length ← Length - 1;
   Done ← TRUE;
   LeafCount ← 1}
 ELSE { -- not a leaf
    Done ← FALSE;
    IF First THEN
      {Child↑ ← Parent↑;
      Child↑.Length ← Child↑.Length + 1};
    {OPEN Child↑;
     Children ← Parent↑.Children;
     IF First THEN
       Seq ← 0
```

```
      ELSE {
         OldNext ← Perm[Length - 1];
        Seq ← 0;
        WHILE Children[Seq] # OldNext
         DO
         Seq ← Seq + 1;
         ENDLOOP;
        Seq ← Seq + 1};
      Perm ← Parent↑.Perm;
      DO
       IF Seq = ProbSize - Length + 1 THEN
        {Done ← TRUE;
         EXIT};
       Perm[Length - 1] ← Children[Seq];
       Cost ← FAdd[Parent↑.Cost,
           InitialData[Perm[Length - 2]][Perm[Length - 1]]];
       IF FComp[Cost,BestInfo.Estimate] = -1 THEN
         EXIT;
       Seq ← Seq + 1;
       ENDLOOP;
       WHILE Seq < ProbSize - Length + 1
        DO
       Children[Seq] ← Children[Seq + 1];
       Seq ← Seq + 1;
       ENDLOOP};
      IF NOT Done THEN
       {SortChildren[Child];
       Child↑.LeafCount ← 0}};
 END;

UseNewInfo: PUBLIC PROCEDURE[Info: InfoType] =
 BEGIN
   IF FComp[Info.Estimate,BestInfo.Estimate] = 1 THEN
     BestInfo.Estimate ← Info.Estimate;
 END;

FirstProb: PUBLIC PROCEDURE[Size: INTEGER] RETURNS[P: ProblemType] =
 BEGIN
  Seq: INTEGER ← 0;

 {OPEN P;
  LeafCount ← 0;
  ProbSize ← Size;
  Length ← 1;
  Perm[0] ← 0;
  Cost ← 0.0;
  BestCost ← Infinity;
  WHILE Seq < ProbSize
   DO
   Children[Seq] ← Seq + 1;
   Seq ← Seq + 1;
   ENDLOOP};
 SortChildren[@P];
 END;
```

```
Combine: PUBLIC PROCEDURE[Child: LONG POINTER TO ProblemType,
                          Parent: LONG POINTER TO ProblemType]
                          RETURNS[Update: BOOLEAN] =
 BEGIN
  OPEN Parent↑;
  LeafCount ← LeafCount + Child↑.LeafCount;
  IF FComp[Child↑.BestCost,BestCost] = -1 THEN
    {BestCost ← Child↑.BestCost;
     BestPerm ← Child↑.BestPerm};
  Update ← FALSE;
 END;

NonTrivial: PUBLIC PROCEDURE[P: ProblemType] RETURNS[BOOLEAN] =
 BEGIN
  RETURN[P.ProbSize - P.Length > 2];
 END;

AppliclInit: PUBLIC PROCEDURE[] =
 BEGIN
  WriteString[Title];
  WriteString[" "];
  NewLine[];
  BestInfo.Estimate ← Infinity;
  InitReals[];
 END;

 ApplicFinish: PUBLIC PROCEDURE[] =
 BEGIN
 END;

AcceptRoot: PUBLIC PROCEDURE[P: ProblemType] =
 BEGIN
  Seq: INTEGER ← 0;

  WriteString["Best path  "L];
  {OPEN P;
   WHILE Seq < ProbSize
    DO
    WriteString[" "L];
    WriteDecimal[BestPerm[Seq]];
    Seq ← Seq + 1;
    ENDLOOP;
  WriteString[" ("L];
  WriteFloat[BestCost];
  WriteString["), "L];
  WriteLongDecimal[LeafCount];
  WriteString[" Leaves Searched"L];
  NewLine[]};
 END;

AcceptAnswer: PUBLIC PROCEDURE[P: ProblemType] =
 BEGIN
  Seq: INTEGER ← 0;
```

```
  WriteString["New Path  "L];
  {OPEN P;
   WHILE Seq < Length
    DO
    WriteString[" "L];
    WriteDecimal[Perm[Seq]];
    Seq ← Seq + 1;
    ENDLOOP;
   WriteString[" ("L];
   WriteFloat[Cost];
   WriteString[") "L];
   NewLine[]};

  END;

 Update: PUBLIC PROCEDURE[Child: LONG POINTER TO ProblemType,
                          Parent: LONG POINTER TO ProblemType]
                          RETURNS[Again: BOOLEAN] =
  BEGIN
   Again ← FALSE;
  END;

 PrintAnswer: PUBLIC PROCEDURE[P: ProblemType] =
  BEGIN
  END;

END.
```

## Appendix C: Execution Results

### C.1 Queens Problem Results

#### Queens - 4

2 4 1 3
3 1 4 2

#### Queens - 5

1 3 5 2 4
1 4 2 5 3
2 4 1 3 5
2 5 3 1 4
3 1 4 2 5
3 5 2 4 1
4 1 3 5 2
4 2 5 3 1
5 2 4 1 3
5 3 1 4 2

#### Queens - 6

2 4 6 1 3 5
3 6 2 5 1 4
4 1 5 2 6 3
5 3 1 6 4 2

#### Queens - 7

1 3 5 7 2 4 6
1 4 7 3 6 2 5
1 5 2 6 3 7 4
1 6 4 2 7 5 3
2 4 1 7 5 3 6
2 4 6 1 3 5 7
2 5 1 4 7 3 6
2 5 3 1 7 4 6
2 5 7 4 1 3 6
2 6 3 7 4 1 5
2 7 5 3 1 6 4
3 1 6 2 5 7 4
3 1 6 4 2 7 5
3 5 7 2 4 6 1
3 6 2 5 1 4 7
3 7 2 4 6 1 5
3 7 4 1 5 2 6
4 1 3 6 2 7 5
4 1 5 2 6 3 7
4 2 7 5 3 1 6
4 6 1 3 5 7 2
4 7 3 6 2 5 1
4 7 5 2 6 1 3
5 1 4 7 3 6 2
5 1 6 4 2 7 3
5 2 6 3 7 4 1
5 3 1 6 4 2 7
5 7 2 4 6 1 3
5 7 2 6 3 1 4
6 1 3 5 7 2 4
6 2 5 1 4 7 3
6 3 1 4 7 5 2
6 3 5 7 1 4 2
6 3 7 4 1 5 2
6 4 2 7 5 3 1
6 4 7 1 3 5 2
7 2 4 6 1 3 5
7 3 6 2 5 1 4
7 4 1 5 2 6 3
7 5 3 1 6 4 2

#### Queens - 8

1 5 8 6 3 7 2 4
1 6 8 3 7 4 2 5
1 7 4 6 8 2 5 3
1 7 5 8 2 4 6 3
2 4 6 8 3 1 7 5
2 5 7 1 3 8 6 4
2 5 7 4 1 8 6 3
2 6 1 7 4 8 3 5
2 6 8 3 1 4 7 5
2 7 3 6 8 5 1 4
2 7 5 8 1 4 6 3
2 8 6 1 3 5 7 4
3 1 7 5 8 2 4 6
3 5 2 8 1 7 4 6
3 5 2 8 6 4 7 1
3 5 7 1 4 2 8 6
3 5 8 4 1 7 2 6
3 6 2 5 8 1 7 4
3 6 2 7 1 4 8 5
3 6 2 7 5 1 8 4
3 6 4 1 8 5 7 2
3 6 4 2 8 5 7 1
3 6 8 1 4 7 5 2
3 6 8 1 5 7 2 4
3 6 8 2 4 1 7 5
3 7 2 8 5 1 4 6
3 7 2 8 6 4 1 5
3 8 4 7 1 6 2 5
4 1 5 8 2 7 3 6
4 1 5 8 6 3 7 2
4 2 5 8 6 1 3 7
4 2 7 3 6 8 1 5
4 2 7 3 6 8 5 1
4 2 7 5 1 8 6 3
4 2 8 5 7 1 3 6
4 2 8 6 1 3 5 7
4 6 1 5 2 8 3 7
4 6 8 2 7 1 3 5
4 6 8 3 1 7 5 2
4 7 1 8 5 2 6 3
4 7 3 8 2 5 1 6

47526138
47531682
48136275
48157263
48531726
51468273
51842736
51863724
52468317
52473861
52617483
52814736
53168247
53172864
53847162
57138642
57142863
57248136
57263148
57263184
57413862
58413627
58417263
61528374
62713584
62714853
63175824
63184275
63185247
63571428
63581427
63724815
63728514
63741825
64158273
64285713
64713528
64718253
68241753
71386425
72418536
72631485
73168524
73825164
74258136
74286135
75316824
82417536
82531746
83162574
84136275

## C.2 Knights Problem Result

### Knights - 3

(1,1) (2,3) (3,1) (1,2) (3,3) (2,1).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2).

### Knights - 4

(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3).
(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2).

(1,1) (2,3) (4,4) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3) (2,1) (4,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2) (3,4) (4,2).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4).
(1,1) (2,3) (4,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (3,4) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (3,4) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3).

(1,1) (2,3) (3,1) (4,3) (2,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4) (3,2) (1,3).
(1,1) (2,3) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (2,4) (1,2) (3,3).
(1,1) (2,3) (3,1) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (2,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (1,2) (2,4) (3,2) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (1,2) (2,4) (3,2) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (2,4).
(1,1) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (1,2) (2,4) (3,2) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (1,2) (2,4) (3,2) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3).
(1,1) (2,3) (3,1) (1,2) (2,4) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,2) (4,4).

(1,1) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (2,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (4,3) (2,4) (3,2) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (4,3) (2,4) (3,2) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3) (3,2) (2,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (4,3) (2,4) (3,2) (1,3) (3,4) (4,2).
(1,1) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (4,3) (2,4) (3,2) (1,3) (2,1) (4,2).
(1,1) (2,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4) (3,2) (1,3).
(1,1) (2,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (1,3) (3,2) (2,4) (4,3) (2,2).
(1,1) (2,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,2) (2,4) (4,3) (2,2) (3,4) (4,2).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,3) (2,4) (3,2) (1,3) (2,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (1,3) (2,1) (3,3).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,2) (2,4).
(1,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (2,1) (1,3) (3,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (2,1) (1,3) (3,2) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (1,3).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (2,1) (1,3) (3,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (2,1) (1,3) (3,2) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (1,3).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (1,2) (3,1).

(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3).
(1,1) (2,3) (4,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (3,4) (1,3) (3,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (1,3).
(1,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (3,4) (1,3) (3,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (3,4) (1,3) (3,2) (2,4) (1,2) (3,1).
(1,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (1,3).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,2) (2,4) (3,2) (1,3) (3,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,4) (3,2) (1,3) (3,4) (2,2).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3) (3,2) (2,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (3,2) (4,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (1,2) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (4,3) (3,1).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3).
(1,1) (2,3) (4,2) (2,1) (1,3) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2).

(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (3,1).

(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (4,4) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2).
(1,1) (3,2) (2,4) (4,3) (3,1) (1,2) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,3) (3,1) (1,2) (3,3) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,1) (3,3) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (3,3) (1,2) (3,1) (2,3) (4,2).

(1,1) (3,2) (2,4) (4,3) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (3,1) (1,2) (3,3).
(1,1) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (3,1).
(1,1) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (1,2) (3,1) (2,3) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (4,3) (2,2) (1,4) (3,3) (1,2) (3,1) (2,3) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (3,1).
(1,1) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (1,2) (3,1) (2,3) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (4,3) (2,2) (4,1) (3,3) (1,2) (3,1) (2,3) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (3,1).
(1,1) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (4,3) (3,1) (2,3) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (1,4) (2,2) (4,3) (3,1) (2,3) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (3,1).
(1,1) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (4,3) (3,1) (2,3) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (4,1) (2,2) (4,3) (3,1) (2,3) (4,2) (2,1) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (2,1) (4,2) (3,4) (2,2) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (4,2) (2,3) (3,1) (4,3) (2,2).
(1,1) (3,2) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,3) (2,1) (1,3) (3,4) (2,2) (4,3) (3,1) (2,3) (4,2).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (3,4) (1,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (1,4) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (2,4) (1,2) (3,1) (4,3) (2,2) (4,1) (3,3) (2,1) (1,3) (3,4) (4,2).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (3,4) (2,2) (1,4) (3,3) (2,1) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (3,4) (2,2) (4,1) (3,3) (2,1) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (1,4) (2,2).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (3,4) (1,3) (2,1) (3,3) (4,1) (2,2).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (2,1) (3,3) (1,4) (2,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (2,1) (3,3) (4,1) (2,2) (3,4) (1,3).
(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (1,4) (3,3).

(1,1) (3,2) (2,4) (1,2) (3,1) (2,3) (4,2) (2,1) (1,3) (3,4) (2,2) (4,1) (3,3).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (4,3) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,4) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (4,1) (2,2) (4,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (4,2) (2,1) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,3) (2,1) (4,2) (2,3) (3,1).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,2) (2,1) (3,3).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,3) (3,1) (2,3) (4,2) (2,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,3) (3,1) (1,2) (3,3) (2,1) (4,2) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (2,1) (4,2) (2,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,2).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (2,1) (4,2) (2,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,2).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (3,4) (4,2) (2,3) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (3,4) (4,2) (2,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,4) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,2).
(1,1) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (3,4) (4,2) (2,3) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (3,4) (4,2) (2,3) (3,1) (1,2) (2,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,4).

(1,1) (3,2) (1,3) (2,1) (3,3) (4,1) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,2).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (3,1) (2,3) (4,2) (3,4) (2,2).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2) (2,4) (4,3) (2,2) (3,4) (4,2) (2,3) (3,1).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2) (3,1) (4,3) (2,2) (3,4) (4,2) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (3,3) (1,2) (3,1) (2,3) (4,2) (3,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,3) (2,4) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (1,4) (3,3) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (2,4) (4,3) (3,1).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (3,4) (2,2) (4,1) (3,3) (1,2) (3,1) (2,3) (4,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (1,4) (2,2).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,4) (1,2) (3,3) (4,1) (2,2).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,2) (1,4) (3,3) (1,2) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (4,3) (2,2) (4,1) (3,3) (1,2) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (1,4) (3,3).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (1,2) (2,4) (4,3) (2,2) (4,1) (3,3).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (1,2) (3,3) (1,4) (2,2) (4,3) (2,4).
(1,1) (3,2) (1,3) (2,1) (4,2) (2,3) (3,1) (1,2) (3,3) (4,1) (2,2) (4,3) (2,4).

336 Leaves .

## C.3 Traveling Salesman Problem Results

Sorted Travel - 4
Best path 0 1 2 3 (1.36954)

Sorted Travel - 5
Best path 0 1 4 2 3 (1.43378)

Sorted Travel - 6
Best path 0 1 4 2 3 5 (1.40272)

Sorted Travel - 7
Best path 0 5 6 2 4 1 3 (1.65287)

Sorted Travel - 8
Best path 0 1 4 7 6 2 3 5 (1.57089)

Sorted Travel - 9
Best path 0 1 3 5 2 4 7 6 8 (2.01829)

Sorted Travel - 10
Best path 0 9 8 1 4 7 6 2 3 5 (2.09491)

Sorted Travel 11
Best path 0 10 1 4 7 6 2 3 5 9 8 (2.3249)

Sorted Travel -12
Best path 0 10 7 4 1 3 5 9 6 2 11 8 (2.08736)

Sorted Travel -13
Best path 0 10 5 6 2 11 8 1 3 9 4 7 12 (1.90502)

Sorted Travel -14
Best path 0 10 1 3 5 9 6 2 11 8 13 4 7 12 (1.87925)

## Appendix D: DIB SOURCE CODES

**File name:** DIBHost.config

```
DIBHost: CONFIGURATION
    IMPORTS String,Stream,NetworkStream,Heap,Tool,TTYSW,Process,
            TTY,System,Put,Exec,FormSW,AddressTranslation,Format
    CONTROL HInterfaceImpl, HostImpl =

    BEGIN
        HInterfaceImpl;
        RealImpl;
        IOImpl;
        TimeMeasureImpl;
        DeweyImpl;
        ApplicImpl;
        HostImpl;
    END.
```

**File name:** DIBNode.config

```
DIBNode: CONFIGURATION
    IMPORTS String,Stream,NetworkStream,Heap,Tool,TTYSW,Process,
            TTY,System,Put,Exec,FormSW,AddressTranslation,
            Format,Time
    CONTROL NInterfaceImpl, NodeImpl =

    BEGIN
        NInterfaceImpl;
        RealImpl;
        NIOImpl;
        TimeMeasureImpl;
        RandomProg;
        DeweyImpl;
        ApplicImpl;
        NodeImpl;
    END.
```

**File name:** DeweyDfs.mesa

```
DeweyDfs : DEFINITIONS =
    BEGIN
        MaxDepth : INTEGER = 25; --depth of recursion allowed
        Comparison : TYPE = {Smaller, Bigger, Equal, Indeterminate};
        Dewey : TYPE = RECORD [
            DNums : ARRAY [1..MaxDepth] OF INTEGER,
            Length : INTEGER];
        DeweyCompare : PROCEDURE[A,B : Dewey]
                                    RETURNS[Answer: Comparison];
        DeweySubsumes : PROCEDURE[Higher,Lower: Dewey]
                                    RETURNS[Answer: BOOLEAN];
        PrintDewey : PROCEDURE[D : Dewey];
     END. --Dewey
```

**File name:** TimemeasureDfs.mesa

```
DIRECTORY
    System USING[GreenwichMeanTime, Pulses];
TimeMeasureDfs: DEFINITIONS =
    BEGIN
        OPEN System;
        TimeMeasure: TYPE = RECORD [
            gmt: GreenwichMeanTime,
            pulses: Pulses];
        Millisecs: TYPE = LONG CARDINAL;

        Get: PROC[] RETURNS [TimeMeasure];
        ElapsedTime: PROC[time: TimeMeasure] RETURNS [Millisecs];
        WriteMillisecs: PROC [m: Millisecs];
    END.
```

**File name:** HInterfaceDfs.mesa

```
DIRECTORY
    InterfaceDfs;

HInterfaceDfs: DEFINITIONS =
    BEGIN
        OPEN InterfaceDfs;
        GetWindowHandle: PROCEDURE [] RETURNS [DataHandle];
        SetUp: PROCEDURE[partit: INTEGER];
        GetInData: PROCEDURE[] RETURNS[MsgBufferPtr];
        Sender: PROCEDURE[outMsgPtr: MsgBufferPtr];
        ClearInQueue: PROCEDURE[];
        <<HostDoIt: PROCEDURE[];>>
    END.
```

**File name:** NInterfaceDfs.mesa

```
DIRECTORY
    InterfaceDfs;

NInterfaceDfs: DEFINITIONS =
    BEGIN
        OPEN InterfaceDfs;
        GetWindowHandle: PROCEDURE [] RETURNS [DataHandle];
        InQueueEmptyCheck: PROCEDURE [] RETURNS [BOOLEAN];
        GetInData: PROCEDURE[] RETURNS[MsgBufferPtr];
        Sender: PROCEDURE[outMsgPtr: MsgBufferPtr];
        ClearInQueue: PROCEDURE[];
    END.
```

```
DIRECTORY
  DeweyDfs,
  IO;

DeweyImpl : PROGRAM
  IMPORTS IO
  EXPORTS DeweyDfs =

  BEGIN  OPEN DeweyDfs,IO;

    PrintDewey : PUBLIC PROCEDURE[D : Dewey] =
      BEGIN
        Seq : INTEGER;

        WriteChar['<];
        {OPEN D;
         Seq ← 1;
         WHILE Seq < = Length
           DO
           WriteDecimal[DNums[Seq]];
           WriteString[". "L];
           Seq ← Seq + 1;
           ENDLOOP};
         WriteString[" > "];
      END;  – – PrintDewey

    DeweyCompare : PUBLIC PROCEDURE[A,B: Dewey] RETURNS[Answer· Comparison] =
    < <Equal means identical, Smaller means A is shorter than
      B (higher in the tree) or to the left if a sibling. > >
      BEGIN
        Seq : INTEGER;

        Answer ← Indeterminate;
        {OPEN A;
         SELECT TRUE FROM
           Length < B.Length = > Answer ← Bigger;
           Length > B.Length = > Answer ← Smaller;
           Length = 0 = > Answer ← Equal;
           ENDCASE = > { – – A.Length = B.Length, non – trivial
                         Seq ← 0;
                         DNums[Length + 1] ← 0;  – – pseudo – data
                         B.DNums[Length + 1] ← 1;  – – pseudo – data
                         WHILE Answer = Indeterminate
                           DO
                           Seq ← Seq + 1,
                           SELECT TRUE FROM
                             DNums[Seq] < B.DNums[Seq] = > Answer ← Smaller,
                             DNums[Seq] > B.DNums[Seq] = > Answer ← Bigger;
                             ENDCASE;
                           ENDLOOP;
                         IF Seq = Length + 1 THEN
                           Answer ← Equal}},

      END;  – – DeweyCompare

    DeweySubsumes : PUBLIC PROCEDURE[Higher,Lower: Dewey]
                RETURNS[Answer: BOOLEAN] =
    < <return True if the Higher is a prefix of Lower > >

      BEGIN
        Seq : INTEGER;

        Answer ← FALSE;
        {OPEN Higher;
         IF Length < = Lower.Length THEN
           {Seq ← 1,
            WHILE Seq < = Length
              DO
              IF DNums[Seq] # Lower DNums[Seq] THEN
                EXIT;
              Seq ← Seq + 1,
              ENDLOOP;
            Answer ← Seq > Length}};
      END;  – – DeweySubsumes

END.  – – DeweyImpl
```

```
DIRECTORY
  Ascii USING[SP],
  IO,
  InterfaceDfs,
  HInterfaceDfs,
  Exec,
  Format,
  FormSW,
  Tool,
  ToolWindow USING [TransitionProcType],
  TTY USING[CharsAvailable, Handle],
  TTYSW USING[GetTTYHandle],
  NetworkStream,
  Stream,
  System USING [NetworkAddress, nullNetworkAddress],
  Process,
  Put USING [LongString, NetworkAddress],
  String USING[Equal],
  Heap USING[systemZone],
  Window USING[Handle],
  AddressTranslation;

HInterfaceImpl: MONITOR
  IMPORTS String,Stream,NetworkStream,Heap,AddressTranslation,Exec,Format,
          FormSW,Tool,Process,Put,IO,TTYSW,TTY < <,HInterfaceDfs > >
  EXPORTS HInterfaceDfs =

  BEGIN
    OPEN InterfaceDfs,HInterfaceDfs,AddressTranslation,IO,
    zone: UNCOUNTED ZONE = Heap.systemZone;
    data: DataHandle ← NIL;
    wh  Window Handle ← NIL;

    dataAvailable  CONDITION,
    receiver: PROCESS ← NIL;
    < <problemSolver: PROCESS ← NIL; > >
    FrontInBuffers: MsgBufferPtr ← NIL;
    RearInBuffers: MsgBufferPtr ← NIL;
    AllNodeList: ARRAY[1..14] OF NodeData ← ALL[[NIL, System.nullNetworkAddress]],
    NodeInUse: NodeInUseType ← ALL[System.nullNetworkAddress],

      - - Command from Form Subwindow

    Assert:  FormSW.ProcType = {Process.Detach[FORK DoAssert]};

    DoAssert:  PROCEDURE[] =
      BEGIN
        IF receiver # NIL THEN
            {Process.Abort[receiver];
             JOIN receiver,
             receiver ← NIL};
        < <IF problemSolver # NIL THEN
            {Process.Abort[problemSolver];
             JOIN problemSolver;
             problemSolver ← NIL}, > >
        data address ← Find[data myName],
        Put.LongString[data.msgSW,data.myName];
        Put.LongString[data.msgSW," "];
        IF data.address # System.nullNetworkAddress THEN
            {Put.NetworkAddress[data.msgSW,data.address,octal];
             receiver ← FORK Receiver[]}
        ELSE Put.LongString[data.msgSW,"This workstation can not be used. Please check the
        user manual."];

      END;

    - - Executive interface procedures

    WindowInit: PROCEDURE[] =
      BEGIN
        wh ← Tool.Create[
                    makeSWsProc: MakeSWs, initialState: default,
                    clientTransition: ClientTransition, name: "DIBHost"
                    cmSection: "DIBHost"];
```



```
      END;

  Null: Exec.ExecProc = {
          Format.Line[Exec.OutputProc[h],
                      "A nothing burger.\n"]};

  Help: Exec.ExecProc = {
          Format.Line[Exec.OutputProc[h],
                      "Use \"DIBHost.~\" to unload the tool.\n"]};

  Unload: Exec.ExecProc = {
            Exec.RemoveCommand[h, "DIBHost.~"];
            IF wh # NIL THEN
                Tool.Destroy[wh]
            ELSE Format.Line
                    [Exec.OutputProc[h], "No DIBHost tool to delete."]};

  - - Tool interface procedures

  ClientTransition: ToolWindow.TransitionProcType = {
      SELECT TRUE FROM
          old = inactive = >
              IF data = NIL THEN data ← zone.NEW[Data ← []],
          new = inactive = > {
              IF receiver # NIL THEN
                  {Process.Abort[receiver];
                   JOIN receiver;
                   receiver ← NIL};
              < <IF problemSolver # NIL THEN
                  {Process.Abort[problemSolver];
                   JOIN problemSolver;
                   problemSolver ← NIL}; > >
              IF data # NIL THEN zone.FREE[@data]};
          ENDCASE};

  MakeSWs. Tool.MakeSWsProc = {
      root: LONG STRING = "DIBHost log"L;
      logName: LONG STRING = "DIBHost.logXXX";
      data.msgSW ← Tool.MakeMsgSW[window: window, lines: 4, h. 48];
      data.formSW ← Tool.MakeFormSW[
                  window: window, formProc: MakeForm];
      Tool.UnusedLogName[unused: logName, root: root];
      data ttySW ← Tool.MakeTTYSW[
                  window. window, name: logName, h: 96]},

  MakeForm  FormSW.ClientItemsProcType = {
      OPEN FormSW;
      nItems: CARDINAL = (FormItems.LAST.ORD – FormItems.FIRST.ORD) + 1;
      items ← AllocateItemDescriptor[nItems];
      items[FormItems.assert.ORD] ← CommandItem[
                  tag: "Assert", place: [6, line0], proc: Assert];
      items]FormItems.iam.ORD] ← StringItem[
                  tag. "I am", inHeap: TRUE, place: [66, line0],
                  string. @data.myName];
      RETURN]items: items, freeDesc: TRUE]};

  GetWindowHandle: PUBLIC PROCEDURE[] RETURNS[DataHandle] =
    BEGIN
      RETURN[data];
    END;


  GetInData. PUBLIC ENTRY PROCEDURE[] RETURNS[MsgBufferPtr] =
    BEGIN
      tmpPtr. MsgBufferPtr ← NIL,

      WHILE FrontInBuffers = NIL OR FrontInBuffers ↑ .status = 0
        DO
        WAIT dataAvailable;
        ENDLOOP;
      tmpPtr ← FrontInBuffers;
      FrontInBuffers ← FrontInBuffers ↑ .next;
      IF FrontInBuffers = NIL THEN
          RearInBuffers ← FrontInBuffers;
```

```
    RETURN[tmpPtr];
  ENO;

ClearInQueue: PUBLIC PROCEOURE[] =
  BEGIN
    next: MsgBufferPtr ← NIL;

    WHILE FrontInBuffers # NIL
      OO
        next ← FrontInBuffers;
        FrontInBuffers ← FrontInBuffers↑.next;
        zone.FREE[@next];
      ENOLOOP;
    RearInBuffers ← FrontInBuffers;
  ENO;

StoreInOata: ENTRY PROCEOURE[tmpPtr: MsgBufferPtr] =
  BEGIN
    IF FrontInBuffers = NIL THEN
        {FrontInBuffers ← tmpPtr;
         RearInBuffers ← FrontInBuffers}
    ELSE {RearInBuffers↑.next ← tmpPtr;
          RearInBuffers ← tmpPtr},
    RearInBuffers↑.status ← 1;
    NOTIFY dataAvailable;
  ENO;

SetUp: PUBLIC PROCEOURE[partit: INTEGER] =
  BEGIN
    tty: TTY.Handle ← TTYSW.GetTTYHandle[data.ttySW];
    realMsg: MACHINE OEPENOENT RECORO [
                        length, maxlength: CAROINAL,
                        text: PACKEO ARRAY[0..10) OF CHARACTER] ←
                            [0, 10, ALL[Ascii.SP]];
    NodeName: LONG STRING ← LOOPHOLE[LONG[@realMsg]];
    stream: Stream.Handle ← NIL;
    Seq: INTEGER ;
    b: MACHINE OEPENOENT RECORO [
        prefix: PACKEO ARRAY [0..0) OF Stream.Byte,
        mchnum: INTEGER,
        usedNode: NodeInUseType];

    Seq ← 0;
    WHILE Seq < = partit
      OO
      WriteString["Please input the name of each node – first is the Host "L];
      NewLine[];
      WHILE TTY.CharsAvailable[tty] = 0
        OO
        Process.Pause[25]
        ENOLOOP;
      NodeName.length ← 0;
      ReadLine[NodeName];
      NodeInUse[Seq] ← Find[NodeName];
      Seq ← Seq + 1;
      ENOLOOP;
    Seq ← 1;
    WHILE Seq < = partit
      OO
      ENABLE
          NetworkStream ConnectionFailed,
              NetworkStream.ConnectionSuspended = > {
                  IF stream # NIL THEN
                      stream.delete[stream];
                      stream ← NIL;
                  WriteString["SetUp – ConnectionFailed or
                  ConnectionSuspended,RETRY..."];
                      WriteOecimal[Seq];
                      LOOP};
      stream ← NetworkStream.Create[
                  remote: NodeInUse[Seq],
                  timeout  60000,
                  classOfService: transactional];
      Stream.SetInputOptions[stream, [terminateOnEndRecord:TRUE]];
      b.mchnum ← Seq;
      b.usedNode ← NodeInUse;
      Stream.SetSST[stream, NodeNameSST];
      Stream.PutBlock[
                  sH: stream,
                  block:[@b.prefix,
                          0, 2 + BytesInNodeInUse],
                  endRecord: TRUE];
      IF NetworkStream.Close[stream] # good THEN
          {stream.delete[stream];
           stream ← NIL;
           WriteString["SetUp – Close is not good, RETRY"];
           NewLine[];
           LOOP};
      stream.delete[stream];
      stream ← NIL;
      Seq ← Seq + 1;
      ENOLOOP;
  ENO;

Find: PROCEOURE[name: LONG STRING]
        RETURNS[addr: System.NetworkAddress ← System.nullNetworkAddress] =
  BEGIN
    Seq: INTEGER ← 1;
    Found: BOOLEAN ← FALSE;
    Equal: BOOLEAN ← FALSE;

    WHILE Seq < = 14 ANO NOT Found
      OO
      Equal ← String.Equal[AllNodeList[Seq].name, name];
      IF Equal THEN
        {addr ← AllNodeList[Seq].addr;
         Found ← TRUE};
      Seq ← Seq + 1;
      ENOLOOP;
  ENO;

Sender: PUBLIC PROCEOURE[outMsgPtr: MsgBufferPtr] =
  BEGIN
    stream: Stream.Handle ← NIL,
    Retry: SIGNAL = COOE,

    BEGIN
      ENABLE BEGIN
                Retry = > {IF stream # NIL THEN
                              {stream.delete[stream],
                               stream ← NIL};
                           WriteString["Sender – Close is not good, RETRY"];
                           RETRY};
                NetworkStream.ConnectionFailed,
                      NetworkStream.ConnectionSuspended = > {
                          IF stream # NIL THEN
                            stream.delete[stream];
                          stream ← NIL;
                          WriteString["Sender – ConnectionFailed or
                          ConnectionSuspended,RETRY..."];
                          WriteOecimal[outMsgPtr↑.dest],
                          NewLine[];
                          RETRY};
             ENO;

      stream ← NetworkStream.Create[
                  remote:  NodeInUse[outMsgPtr↑.dest],
                  timeout. 200000,
                  classOfService: transactional];
      Stream.SetInputOptions[stream, [terminateOnEndRecord: TRUE]];
      Stream.SetSST[stream, MessageSST];
      Stream.PutBlock[
                  sH: stream,
                  block:[@outMsgPtr↑.prefix,
                          0, BytesInMsgBuffer],
                  endRecord: TRUE];
      IF NetworkStream.Close[stream] # good THEN
          SIGNAL Retry,
```

```
        stream.delete[stream];
        stream ← NIL;
        zone.FREE[@outMsgPtr];
      END;
    END;


  Receiver: PROCEDURE[] =
    BEGIN
      stream: Stream.Handle ← NIL;
      lIstener: NetworkStream.LIstenerHandle ←
            NetworkStream.CreateListener[data.address[;

      DO
        ENABLE BEGIN
          ABORTED = > EXIT;
          NetworkStream.ConnectionFailed,
            NetworkStream.ConnectionSuspended = > {
              IF stream # NIL THEN
                  {stream.delete[stream];
                   stream ← NIL};
              WriteString["Receiver – ConnectionFailed or ConnectionSuspended, LOOP"L[;
              LOOP};
          END «SIGNAL ENABLEs»;

        [] ← NetworkStream.Listen[listenerH: listener];
        stream ← NetworkStream.ApproveConnection[
                    listenerH: listener,
                    classOfService: transactional];
        Process.Detach[FORK ReceiveInstance[stream]];
      ENDLOOP,
      NetworkStream.DeleteListener[listener !ABORTED = > CONTINUE];
    END;

  ReceiveInstance: PROCEDURE [stream: Stream.Handle] =
    BEGIN
      tmpPtr: MsgBufferPtr ← NIL;
      EmptyBlock: MACHINE DEPENDENT RECORD[
          prefix: PACKED ARRAY [0..0) OF Stream.Byte,
          pseudo: MsgBuffer];
      size: CARDINAL ← 0;
      why: Stream.CompletionCode ← normal;
      sst, dummy: Stream.SubSequenceType ← 0;

      BEGIN
        ENABLE BEGIN
                NetworkStream.ConnectionFailed,
                NetworkStream.ConnectionSuspended = > {
                    IF stream # NIL THEN
                        {stream.delete[stream],
                         stream ← NIL};
                    WriteString["Receiver1 – ConnectionFailed or ConnectionSuspended,
                    CONTINUE"L],
                    CONTINUE};
              END;
        stream.setTimeout[sH: stream, waitTime: 200000];
        Stream.SetInputOptions[stream, [terminateOnEndRecord: TRUE]],
        tmpPtr ← zone.NEW[MsgBuffer],
        size ← 0; dummy ← 0; why ← sstChange,
        WHILE size = 0
          DO
          IF why = sstChange THEN sst ← dummy,
          [size, why, dummy] ← Stream.GetBlock[
                                    stream,
                                    [@tmpPtr↑.prefix,0,BytesInMsgBuffer]
                                     ! Stream.TimeOut = > RESUME]
          ENDLOOP;
        IF (why # endRecord) OR (size < BytesInMsgBuffer) THEN
            {zone.FREE[@tmpPtr];
             GO TO Exit};
        StoreInData[tmpPtr[;

        [size, why, sst] ← Stream.GetBlock[
                                    stream,
                                    [@EmptyBlock.prefix,0,BytesInMsgBuffer]
                                    ! Stream.TimeOut = > RESUME];
        IF (size # 0) OR (sst # NetworkStream.closeSST) THEN
            GO TO Exit;
        IF NetworkStream.CloseReply[stream] # good THEN
            GO TO Exit;

        EXITS Exit = > NULL;
      END;
      IF stream # NIL THEN
          {stream.delete[stream];
           stream ← NIL};
    END;
    – – main
  Exec.AddCommand[name: "DIBHost.~",
                  proc: Null, help: Help, unload: Unload];
  WindowInit[];
  Process.EnableAborts[@dataAvailable];
  AllNodeList[1].name ← "ALVIS";
  AllNodeList[1].addr ←
            StringToNetworkAddress["5772.25200121351.71525",NIL,NIL].addr;
  AllNodeList[2].name ← "LIT";
  AllNodeList[2].addr ←
            StringToNetworkAddress["5772.25200121602.16756",NIL,NIL].addr;
  AllNodeList[3].name ← "FJALAR";
  AllNodeList[3].addr ←
            StringToNetworkAddress["5772 25200121213.13160",NIL,NIL].addr;
  AllNodeList[4].name ← "ANDVARI";
  AllNodeList[4].addr ←
            StringToNetworkAddress["5771.25200122706.22011",NIL,NIL].addr;
  AllNodeList[5].name ← "GALAR";
  AllNodeList[5].addr ←
            StringToNetworkAddress["5772.25200121527 120223",NIL,NIL].addr;
  AllNodeList[6].name ← "BROKK";
  AllNodeList[6].addr ←
            StringToNetworkAddress["5771.25200115272.26065",NIL,NIL].addr;
  AllNodeList[7].name ← "MODSOGNIR";
  AllNodeList[7].addr ←
            StringToNetworkAddress["5771.25200120764.65150",NIL,NIL].addr;
  AllNodeList[8].name ← "DURIN";
  AllNodeList[8[.addr ←
            StringToNetworkAddress["5771.25200113474 50653",NIL,NIL].addr;
  AllNodeList[9].name ← "EITRI";
  AllNodeList[9].addr ←
            StringToNetworkAddress["5771.25200123657.35367",NIL,NIL].addr;

END.
```

```
DIRECTORY
  ApplicDfs USING[ProblemType,InfoType],
  DeweyDfs USING[Dewey],
  TimeMeasureDfs;

HostDfs: DEFINITIONS =

  BEGIN
    MaxProbSize: INTEGER = 20;
    MaxRedundancy: INTEGER = 5;
    TimeMeasures: INTEGER = 4;
    CommTime: INTEGER = 0;
    TossTime: INTEGER = 1;
    TermTime: INTEGER = 2;
    IdleTime: INTEGER = 3;
    AnswerSpecification: TYPE = {Full,Count};
    StatusSpecification: TYPE = {BatchDone,AnAnswer,
                                 Quitting,Reporting,PleaseStop};
    TimeType: TYPE = ARRAY[0..TimeMeasures + MaxRedundancy] OF
                       TimeMeasureDfs.Millisecs;
                       << calculating, waiting for work/helping peer,
                          printing messages on console, waiting for
                          termination message. >>
    MessageType: TYPE = {Request,Work,GlobalInfo,Result,Terminate,Synch,
                         Repeating,Updating};
    MessageClass: TYPE = {AnswerMessage,ProblemMessage,InterNodeMessage};
    Message: TYPE = RECORD[
      specifics: SELECT type: MessageClass FROM
        AnswerMessage => [
          Status: StatusSpecification,
          Prob: ApplicDfs.ProblemType,
          Times: TimeType,
          Sequence: LONG INTEGER,
          AnswerDewey: DeweyDfs.Dewey],
        ProblemMessage => ]
          Kind  INTEGER, -- 0 means no work; 1 means here is a problem
          Answer: AnswerSpecification,
          Prob: ApplicDfs.ProblemType,
          PartSize: INTEGER, -- partition size
          WorkFraction. INTEGER, << how much to give away when asked;
                                    0 means give away none; 10 means give away
                                    all work; negative means give away one
                                    problem (if possible) >>
          NumHelp: INTEGER, -- how many to ask when out of work
          DebugLevel: INTEGER], << higher numbers are more verbose; 0 is
                                   silent >>
        InterNodeMessage => [
          Kind: MessageType,
          Asker: INTEGER,
          Child: ApplicDfs.ProblemType,
          ChildDewey: DeweyDfs Dewey,
          Count: INTEGER,
          SomeInfo: ApplicDfs.InfoType,
          Redundancy: INTEGER],
        ENDCASE];

  END. -- HostDfs
```

```
DIRECTORY
  IO,
  DeweyDfs,
  ApplicDfs,
  HostDfs,
  InterfaceDfs,
  HInterfaceDfs,
  TimeMeasureDfs USING[TimeMeasure, Millisecs, Get, WriteMillisecs,
                       ElapsedTime],
  Heap USING[systemZone],
  Process;

HostImpl: PROGRAM
< < Host end of distributed backtracking > >
  IMPORTS IO,DeweyDfs,ApplicDfs,HInterfaceDfs,TimeMeasureDfs,
          Process,Heap
  < <EXPORTS HInterfaceDfs> > =
  BEGIN
    OPEN IO,DeweyDfs,ApplicDfs,HostDfs,InterfaceDfs,HInterfaceDfs,
         TimeMeasureDfs;

    zone: UNCOUNTED ZONE = Heap.systemZone,
    data: DataHandle ← NIL,


    MaxPartitionSize: INTEGER = 7;
    Statistic: TYPE = RECORD[
      Time: TimeType,
      Sequence: LONG INTEGER];

    JobStopped: SIGNAL = CODE;
    outMsgPtr: MsgBufferPtr ← NIL;
    AllStats: ARRAY[1..MaxPartitionSize] OF Statistic ← ALL[];
    PartitionSize: INTEGER ← 1;
    Style: AnswerSpecification ← Count;
    WorkFract: INTEGER ← 5;
    NumHelpers. INTEGER ← 1;
    Debug: INTEGER ← 0;


    HSendProblem: PROCEDURE[P: LONG POINTER TO ProblemType,
                            St: AnswerSpecification] =
    < <give a problem to node 1 and 'ask someone' to the other nodes  > >
      BEGIN
        WhereTo: INTEGER ← 1;
        a: ProblemMessage Message;


        WHILE WhereTo < = PartitionSize
          DO
          outMsgPtr ← zone.NEW[MsgBuffer];
          a.Prob ← P↑,
          IF WhereTo = 1 THEN
              a.Kind ← 1
          ELSE a.Kind ← 0;
          a.Answer ← St;
          a.PartSize ← PartitionSize,
          a.WorkFraction ← WorkFract;
          a.NumHelp ← NumHelpers;
          a.DebugLevel ← Debug,
          outMsgPtr↑ message ← a;
          outMsgPtr↑.source ← 0;
          outMsgPtr↑.dest ← WhereTo,
          Sender[outMsgPtr],
          WhereTo ← WhereTo + 1;
          ENDLOOP;
      END; – – HSendProblem

    ClearStats: PROCEDURE[] =
      BEGIN
        Machine: INTEGER ← 1;
        Category: INTEGER;

        WHILE Machine < = PartitionSize
          DO
          {OPEN AllStats[Machine];
           Sequence ← 0;
           Category ← 0;
           WHILE Category < = TimeMeasures + MaxRedundancy
             DO
             Time[Category] ← 0;
             Category ← Category + 1;
             ENDLOOP};
          Machine ← Machine + 1;
          ENDLOOP;
      END; – – ClearStats

    PrintStats: PROCEDURE[] =
      BEGIN
        Machine, Category: INTEGER;
        AllTotal: Statistic;

        AddRow: PROCEDURE[A: Statistic, B: LONG POINTER TO Statistic] =
          BEGIN
            Category: INTEGER;

            {OPEN B↑;
             Category ← 0;
             WHILE Category < = TimeMeasures + MaxRedundancy
               DO
               Time[Category] ← Time[Category] + A.Time[Category];
               Category ← Category + 1
               ENDLOOP};
          END; – – AddRow

        PrintRow: PROCEDURE[Stat: Statistic] =
          BEGIN
            RowTotal: Millisecs ← 0;
            Category: INTEGER ← 0;

            {OPEN Stat;
             WHILE Category < = TimeMeasures + MaxRedundancy – 2
               DO
               SELECT Category FROM
                 CommTime = > WriteString["Comm "];
                 TermTime = > WriteString["Term "];
                 TossTime = > WriteString["Toss "];
                 IdleTime = > WriteString["Idle "],
                 TimeMeasures = > WriteString["Work "]
                 ENDCASE;
               WriteMillisecs[Time[Category]];
               WriteString["  "];
               RowTotal ← RowTotal + Time[Category];
               Category ← Category + 1;
               ENDLOOP;
             WriteString[" = "];
             WriteMillisecs[RowTotal];
             NewLine[]};
          END; – – PrintRow

        – – AllTotal ← 0
        Category ← 0;
        WHILE Category < = TimeMeasures + MaxRedundancy
          DO
          AllTotal.Time[Category] ← 0;
          Category ← Category + 1;
          ENDLOOP;
        Machine ← 1;
        WHILE Machine < = PartitionSize
          DO
          WriteChar['#];
          WriteDecimal[Machine];
          WriteChar[' ];
          PrintRow[AllStats[Machine]];
          AddRow[AllStats[Machine], @AllTotal];
          Machine ← Machine + 1;
          ENDLOOP;
        NewLine[];
```

```
    WriteString[" Total "];
    PrintRow[AllTotal];
    NewLine[];
    -- AllTotal ← AllTotal / PartitionSize
    Category ← 0;
    WHILE Category < = TimeMeasures + MaxRedundancy
      DO
      AllTotal.Time[Category] ← AllTotal.Time[Category] /
        PartitionSize;
      Category ← Category + 1;
      ENDLOOP;
    WriteString[" Average "];
    PrintRow[AllTotal];
    NewLine[]; NewLine[];
  END; -- PrintStats

HWaitDone: PROCEDURE[TM: TimeMeasure]
            RETURNS[Who: INTEGER, Timeused: Millisecs] =
<<wait until a child says it's finished, and return who said that >>
  BEGIN
    Done: BOOLEAN ← FALSE;
    inMsgPtr: MsgBufferPtr ← NIL;

    WHILE NOT Done
      DO
      inMsgPtr ← GetInData[];
      Who ← inMsgPtr↑.source;
      WITH a: inMsgPtr↑.message SELECT FROM
        AnswerMessage = > {
          IF (Who NOT IN [1..7]) OR
                a.Sequence # AllStats[Who].Sequence THEN
            {WriteString["{bad from "];
             WriteDecimal[Who];
             WriteString[" ; type "];
             WriteDecimal[LOOPHOLE[a.Status]];
             WriteString[" ; expected "];
             WriteLongDecimal[AllStats[Who].Sequence];
             WriteString[" got "];
             WriteLongDecimal[a.Sequence];
             NewLine[]}
          ELSE
            {AllStats[Who].Sequence ← AllStats[Who].Sequence + 1;
             SELECT a.Status FROM
              PleaseStop = > {Done ← TRUE;
                              ApplicFinish[]},
              BatchDone = > {IF Debug > = 2 THEN
                              {WriteChar['{],
                               WriteDecimal[Who];
                               WriteChar[',];
                               WriteLongDecimal[a.Times[CommTime]];
                               WriteString["} "]},
                             AcceptRoot[a.Prob]},
              Reporting = > {IF Debug > = 1 THEN
                              {WriteChar['{];
                               WriteDecimal[Who];
                               WriteString["} "],
                               PrintDewey[a.AnswerDewey],
                               NewLine[]}},
              AnAnswer = > AcceptAnswer[a.Prob];
              Quitting = > {IF Who = 1 THEN
                              Timeused ← ElapsedTime[TM];
                            IF Debug > = 1 THEN
                              {WriteChar['{];
                               WriteDecimal[Who];
                               WriteChar[',];
                               WriteLongDecimal[a.Times[CommTime]],
                               WriteChar['}];
                               NewLine[]};
                            AllStats[Who].Time ← a.Times};
              ENDCASE}};
        ENDCASE;
      zone.FREE[@inMsgPtr];
      IF Done THEN
        EXIT;
      ENDLOOP;
  END; -- HWaitDone

HostDoIt: PROCEDURE[] =
  BEGIN
    Size: INTEGER;
    FProb: ProblemType;
    Who : INTEGER;
    BeginTime: TimeMeasure;
    Timeused: Millisecs;
    ch  CHARACTER;

    BEGIN
        ENABLE
            ABORTED = > CONTINUE;

    WHILE data = NIL
      DO
      Process.Pause[30];
      data ← GetWindowHandle[];
      ENDLOOP;
    WriteString["Please input Node Number."L],
    NewLine[];
    PartitionSize ← ReadDecimal[];
    NewLine[];
    SetUp[PartitionSize];
    NewLine[];
    DO
      ENABLE
          JobStopped = > EXIT;

      DO
        WriteString["c: count reporting, not every answer"L];
        NewLine[];
        WriteString["d: set debug level"L];
        NewLine[];
        WriteString["f: full reporting of every answer"L];
        NewLine[];
        WriteString["h. set number of helpers when out of work"L],
        NewLine[];
        WriteString["n: new problem"L];
        NewLine[];
        WriteString["q: quit"L];
        NewLine[];
        NewLine[];
        WriteString["Please input Selection."L];
        NewLine[];
        ch ← ReadChar[];
        NewLine[];
        IF (ch # ' ) AND (ch # '\n) THEN EXIT;
      ENDLOOP;

      SELECT ch FROM
        'q = > SIGNAL JobStopped;
        'f = > Style ← Full,
        'd = > {WriteString["Debug level ( "L];
                WriteDecimal[Debug];
                WriteString[") = "L],
                Debug ← ReadDecimal[];
                NewLine[]};
        'h = > {WriteString["New number of helpers ( "L];
                WriteDecimal[NumHelpers];
                WriteString[") = "L];
                NumHelpers ← ReadDecimal[];
                NewLine[]};
        'c = > Style ← Count;
        'n = > {ApplicInit[],
                WriteString["Please input Problem Size."L];
                NewLine[],
                Size ← ReadDecimal[];
                NewLine[];
                FProb ← FirstProb[Size];
                HSendProblem[@FProb, Style];
                ClearStats[];
```

```
                    BeginTime ← Get[];
                    WriteString["Waiting for job done."L];
                    NewLine[];
                    [Who, Timeused] ← HWaitDone[BeginTime];
                    NewLine[];
                    WriteString["Elapsed Time  "L];
                    WriteMillisecs[Timeused];
                    NewLine[];
                    PrintStats[];
                    ClearInQueue[]};
            ENDCASE;
        ENDLOOP;
        END;
    END; – – HostDoIt


– – main

Process.Detach[FORK HostDoIt[]];

END. – – HostImpl
```

```
DIRECTORY
   HostDfs USING[Message],
   Stream USING [Byte, SubSequenceType],
   System USING [NetworkAddress, nullNetworkAddress],
   Window USING[Handle];

IrinterfaceDfs: DEFINITIONS =

   BEGIN
      FormItems: TYPE = {assert, iam};
      DataHandle: TYPE = LONG POINTER TO Data;
      Data: TYPE = RECORD [
                  msgSW: Window.Handle ← NIL,
                  formSW: Window.Handle ← NIL,
                  ttySW: Window.Handle ← NIL,
                  address: System.NetworkAddress ← System nullNetworkAddress,
                  nodeNumber: INTEGER ← 0,
                  myName: LONG STRING ← NIL];

      MsgBufferPtr: TYPE = LONG POINTER TO MsgBuffer;
      MsgBuffer: TYPE = MACHINE DEPENDENT RECORD[
                           prefix: PACKED ARRAY [0..0) OF Stream.Byte,
                           status: INTEGER ← 0,
                           source: INTEGER,
                           dest: INTEGER,
                           message: HostDfs.Message,
                           next: MsgBufferPtr ← NIL];
      NodeData: TYPE = RECORD [
                     name: LONG STRING,
                     addr: System.NetworkAddress];
      NodeInUseType: TYPE = ARRAY[0..13] OF System.NetworkAddress;
      BytesinMsgBuffer. CARDINAL = 2 * SIZE[MsgBuffer];
      BytesinNodeinUse: CARDINAL = 2 * SIZE[NodeInUseType];
      NodeNameSST. Stream.SubSequenceType = 1,
      MessageSST : Stream.SubSequenceType = 2,

   END.
```

```
IO: DEFINITIONS =

  BEGIN

    SetEcho: PROCEDURE [new: BOOLEAN] RETURNS [old: BOOLEAN];
      «SetEcho changes whether or not you want to echo characters for the
        string – /line – oriented Read procedures.  The value returned is the
        state of echoing.»

    ReadChar: PROCEDURE [] RETURNS [CHARACTER];
    «ReadChar returns the next character typed by the user in this
      little window.»

    ReadLine: PROCEDURE [s: LONG STRING];

    Rubout: SIGNAL [];
      «Rubout indicates that the user typed the DEL/RUBOUT key
      during execution of ReadEditedString, ReadID, or ReadLine.»

    LineOverflow: SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING];
      «LineOverflow indicates that the string supplied to
        ReadEditedString, ReadID, or ReadLine was not sufficiently
        long to hold all the characters received.  A longer string
        (with contents of old s copied in) is typically returned »

    ReadDecimal: PROCEDURE [] RETURNS [INTEGER];
      «ReadDecimal uses ReadID to get a string, and tries to convert
        it to an integer, assuming the string is a signed decimal
        number.  If this conversion fails, the SIGNAL
                        String.InvalidNumber
        is raised.»

      WriteChar: PROCEDURE [c: CHARACTER];
        «WriteChar prints the character at the cursor position in this
          little window.»

      WriteString: PROCEDURE [s: LONG STRING];
        «WriteString prints the string at the cursor position in this
          little window.»

      WriteDecimal: PROCEDURE [n: INTEGER];
        «WriteDecimal prints the given integer as a decimal number,
          written as characters.»

      WriteLongDecimal: PROCEDURE [n: LONG INTEGER];
      WriteFloat: PROCEDURE [n: REAL];

      NewLine: PROCEDURE [];
        «NewLine Prints a new – line character»
  END.  – – IO
```

```
DIRECTORY
   IO,
   InterfaceDfs,
   HInterfaceDfs,
   Real,
   TTY USING [Handle, SetEcho, GetChar, GetLine, LineOverflow, Rubout, GetDecimal,
    PutChar, PutString, PutDecimal, PutLongDecimal,PutCR],
   TTYSW USING[GetTTYHandle];

IOimpl: PROGRAM
    IMPORTS TTY,TTYSW,Real,HInterfaceDfs
    EXPORTS IO =

   BEGIN
      data: InterfaceDfs.DataHandle ← NIL;
      ttyWindow: TTY.Handle;

      SetEcho: PUBLIC PROCEDURE [new: BOOLEAN] RETURNS [old: BOOLEAN] =
         BEGIN
            old ← SELECT TTY SetEcho[h: ttyWindow, new: IF new THEN plain ELSE none]
                  FROM
                     none = > FALSE,
                     plain = > TRUE,
                     ENDCASE = > ERROR
         END «SetEcho»;

      ReadChar: PUBLIC PROCEDURE [] RETURNS [CHARACTER] =
                        {RETURN[TTY.GetChar[ttyWindow]]};

      ReadLine: PUBLIC PROCEDURE [s: LONG STRING] =
         BEGIN
               TTY.GetLine[ttyWindow,s !
                                                TTY.LineOverflow = >
                                                   {RESUME[SIGNAL LineOverflow[s]]};
                                                TTY.Rubout = > {      SIGNAL Rubout[];
                                                                      RESUME
                                                               }
                           ]
         END;

      Rubout: PUBLIC SIGNAL [] = CODE;


      LineOverflow: PUBLIC SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING] = CODE;

      ReadDecimal: PUBLIC PROCEDURE [] RETURNS [INTEGER] =
         BEGIN
            RETURN[TTY.GetDecimal[ttyWindow !
                                                TTY.LineOverflow = >
                                                   {RESUME[SIGNAL LineOverflow[s]]};
                                                TTY.Rubout = > {      SIGNAL Rubout[];
                                                                      RESUME
                                                               }
                                       ]]
         END «ReadDecimal»;

      WriteChar: PUBLIC PROCEDURE [c: CHARACTER] = {TTY.PutChar[ttyWindow,c]};

      WriteString: PUBLIC PROCEDURE [s: LONG STRING] =
      {TTY.PutString[ttyWindow,s]};

      WriteDecimal: PUBLIC PROCEDURE [n:INTEGER] =
                              {TTY.PutDecimal[ttyWindow,n]};

      WriteLongDecimal:PUBLIC PROCEDURE [n:LONG INTEGER] =
                              {TTY.PutLongDecimal[ttyWindow,n]};

      WriteFloat: PUBLIC PROCEDURE[n: REAL] =
                  {Real.WriteReal[WriteChar,
                                    n,Real.DefaultSinglePrecision ,FALSE]};

      NewLine: PUBLIC PROCEDURE [] = {TTY.PutCR[ttyWindow]},

   -- main
   data ← HInterfaceDfs.GetWindowHandle[];
   ttyWindow ← TTYSW.GetTTYHandle[data.ttySW];

END «IOimpl».
```

```
DIRECTORY
  IO,
  InterfaceDfs,
  NInterfaceDfs,
  Exec,
  Format,
  FormSW,
  Tool,
  ToolWindow USING [TransitionProcType],
  NetworkStream,
  Stream,
  System USING [NetworkAddress, nullNetworkAddress],
  Process,
  Put,
  String USING[Equal],
  Heap USING[systemZone],
  Window USING[Handle],
  AddressTranslation;

NInterfaceImpl: MONITOR
  IMPORTS String,Stream,NetworkStream,Heap,AddressTranslation,Exec,Format,
          FormSW,Tool,Process,Put,IO
  EXPORTS NInterfaceDfs =

  BEGIN
    OPEN IO,InterfaceDfs,NInterfaceOfs,AddressTranslation;
    zone: UNCOUNTED ZONE = Heap.systemZone,
    data: DataHandle ← NIL;
    wh: Window.Handle ← NIL;

    receiver: PROCESS ← NIL,
    FrontInBuffers: MsgBufferPtr ← NIL;
    RearInBuffers: MsgBufferPtr ← NIL;
    AllNodeList: ARRAY[1..14] OF NodeData ← ALL[[NIL, System.nullNetworkAddress]],
    NodeInUse: NodeInUseType ← ALL[System.nullNetworkAddress];
    dataAvailable: CONDITION;

      -- Command from Form Subwindow

    Assert: FormSW.ProcType = {Process.Detach[FORK DoAssert]};

    DoAssert: PROCEDURE[] =
      BEGIN
        IF receiver # NIL THEN
            {p: PROCESS = receiver;
             Process.Abort[p];
             JOIN p};
        data.address ← Find[data.myName];
        Put.LongString[data.msgSW,data.myName];
        Put.LongString[data.msgSW," "];
        IF data.address # System.nullNetworkAddress THEN
            {Put.NetworkAddress[data.msgSW,data.address,octal];
             receiver ← FORK Receiver[]}
        ELSE Put.LongString[data.msgSW,"This workstation can not be used. Please check the
        user manual."];
      END;

    -- Executive interface procedures

    WindowInit: PROCEDURE[] =
      BEGIN
        wh ← Tool.Create[
                    makeSWsProc: MakeSWs, initialState: default,
                    clientTransition: ClientTransition, name: "DIBNode"
                    cmSection: "DIBNode"];
      END;

    Null: Exec.ExecProc = {
            Format.Line[Exec.OutputProc[h],
                  "A nothing burger.\n"]};

    Help: Exec.ExecProc = {
            Format.Line[Exec.OutputProc[h],
                  "Use \"DIBNode.~\" to unload the tool.\n"]};

    Unload: Exec.ExecProc = {
            Exec.RemoveCommand[h, "DIBNode.~"];
            IF wh # NIL THEN
                Tool.Destroy[wh]
            ELSE Format.Line
                    [Exec.OutputProc[h], "No DIBNode tool to delete."]};

    -- Tool interface procedures

    ClientTransition: ToolWindow.TransitionProcType = {
      SELECT TRUE FROM
        old = inactive = >
            IF data = NIL THEN data ← zone.NEW[Data ← []];
        new = inactive = > {
            IF receiver # NIL THEN {
                p: PROCESS = receiver;
                Process.Abort[p];
                JOIN p};
            IF data # NIL THEN zone.FREE[@data]};
        ENDCASE};

    MakeSWs: Tool.MakeSWsProc = {
        root: LONG STRING = "DIBNode.log"L;
        logName: LONG STRING = "DIBNode.logXXX";
        data.msgSW ← Tool.MakeMsgSW[window: window, lines: 4, h: 48];
        data.formSW ← Tool.MakeFormSW[
                    window: window, formProc: MakeForm];
        Tool.UnusedLogName[unused: logName, root: root];
        data.ttySW ← Tool.MakeTTYSW[
                window: window, name: logName, h: 96];
        < <sender ← FORK Sender[] > >};

    MakeForm: FormSW.ClientItemsProcType = {
        OPEN FormSW;
        nItems: CARDINAL = (FormItems.LAST.ORD – FormItems.FIRST.ORD) + 1;
        items ← AllocateItemDescriptor[nItems];
        items[FormItems.assert.ORD] ← CommandItem[
                tag: "Assert", place: [6, line0], proc. Assert];
        items[FormItems.iam.ORD] ← StringItem[
                tag: "I am", inHeap: TRUE, place: [66, line0],
                string: @data.myName];
        RETURN[items: items, freeDesc: TRUE]};

    GetWindowHandle: PUBLIC PROCEDURE[] RETURNS[DataHandle] =
      BEGIN
        RETURN[data];
      END;

    InQueueEmptyCheck: PUBLIC PROCEDURE[] RETURNS[BOOLEAN] =
      BEGIN
        IF FrontInBuffers = NIL THEN
          RETURN[TRUE]
        ELSE RETURN[FALSE];
      END;

    GetInData: PUBLIC ENTRY PROCEDURE[] RETURNS[MsgBufferPtr] =
      BEGIN
        tmpPtr: MsgBufferPtr ← NIL;

        WHILE FrontInBuffers = NIL OR FrontInBuffers↑.status = 0
          DO
          WAIT dataAvailable;
          ENDLOOP;
        tmpPtr ← FrontInBuffers;
        FrontInBuffers ← FrontInBuffers↑.next;
        IF FrontInBuffers = NIL THEN
            RearInBuffers ← FrontInBuffers;
        RETURN[tmpPtr];
      END;

    ClearInQueue: PUBLIC PROCEDURE[] =
      BEGIN
        next. MsgBufferPtr ← NIL;
```

```
    WHILE FrontInBuffers # NIL
      DO
        next ← FrontInBuffers;
        FrontInBuffers ← FrontInBuffers↑.next;
        zone.FREE[@next[;
      ENDLOOP;
    RearInBuffers ← FrontInBuffers;
  END;

StoreInData: ENTRY PROCEDURE[tmpPtr: MsgBufferPtr] =
  BEGIN
    IF FrontInBuffers = NIL THEN
        {FrontInBuffers ← tmpPtr;
         RearInBuffers ← FrontInBuffers}
    ELSE {RearInBuffers↑.next ← tmpPtr;
          RearInBuffers ← tmpPtr};
    RearInBuffers↑.status ← 1;
    NOTIFY dataAvailable;
  END;

Find: PROCEOURE[name: LONG STRING]
      RETURNS[addr: System.NetworkAddress ← System nullNetworkAddress[ =
  BEGIN
    Seq: INTEGER ← 1;
    Found: BOOLEAN ← FALSE;
    Equal: BOOLEAN ← FALSE;

    WHILE Seq < = 14 AND NOT Found
      DO
      Equal ← String.Equal[AllNodeList[Seq].name, name];
      IF Equal THEN
        {addr ← AllNodeList[Seq].addr;
         Found ← TRUE};
      Seq ← Seq + 1;
      ENOLOOP,
   END;

Sender: PUBLIC PROCEDURE[outMsgPtr: MsgBufferPtr] =
  BEGIN
    stream: Stream.Handle ← NIL;
    Retry: SIGNAL = CODE;

    BEGIN
      ENABLE BEGIN
                Retry = > {IF stream # NIL THEN
                             {stream.delete[stream];
                              stream ← NIL};
                           WriteString["Sender – Close is not good, RETRY. "L];
                           WriteDecimal[outMsgPtr↑.dest],NewLine[ ];
                           RETRY};
                NetworkStream.ConnectionFailed = > {
                           IF stream # NIL THEN
                             {stream.delete[stream];
                              stream ← NIL};
                           WriteString["Sender – ConnectionFailed,"L],
                           IF outMsgPtr↑.dest = 0 THEN
                             {WriteString["RETRY . 0"L];
                              RETRY}
                           ELSE {WriteString["ByPass "L];
                                 WriteOecimal[outMsgPtr↑.dest]},
                           NewLine[ ];
                           CONTINUE};
                NetworkStream ConnectionSuspended = > {
                           IF stream # NIL THEN
                             {stream.delete[stream];
                              stream ← NIL},
                           WriteString["Sender – ConnectionSuspended,RETRY..."L];
                           WriteOecimal[outMsgPtr↑ dest];NewLine[ ];
                           RETRY};
              ENO,

      stream ← NetworkStream.Create[
                  remote: NodeInUse[outMsgPtr↑.dest],
                  timeout: 200000,
                  classOfServIce: transactional];
      Stream.SetInputOptions[stream, [terminateOnEndRecord: TRUE]];
      Stream.SetSST[stream, MessageSST];
      Stream.PutBlock[
                  sH: stream,
                  block:[@outMsgPtr↑.prefix,
                         0, BytesInMsgBuffer[,
                  endRecord: TRUE];
      IF NetworkStream.Close[stream] # good THEN
          SIGNAL Retry;
      stream.delete[stream];
      stream ← NIL;
      zone.FREE[@outMsgPtr];
    END;
END;

Receiver: PROCEDURE[] =
  BEGIN
    b: MACHINE DEPENDENT RECORD [
          prefix: PACKEO ARRAY [0..0) OF Stream.Byte,
          mchnum: INTEGER,
          usedNode  NodeInUseType];
    EmptyBlock: MACHINE OEPENOENT RECORD[
          prefix: PACKEO ARRAY [0..0) OF Stream.Byte,
          pseudo. MsgBuffer];
    Retry: SIGNAL = CODE;
    size: CARDINAL ← 0;
    why: Stream.CompletionCode ← normal;
    sst, dummy: Stream.SubSequenceType ← 0;
    stream: Stream.Handle ← NIL;
    listener: NetworkStream ListenerHandle ←
          NetworkStream CreateListener[data.address],

    BEGIN
      ENABLE BEGIN
                ABORTED = > CONTINUE;
                Retry = > {IF stream # NIL THEN
                             {stream.delete[stream];
                              stream ← NIL};
                           RETRY};
                NetworkStream.ConnectionFailed,
                NetworkStream.ConnectionSuspended = > {
                     IF stream # NIL THEN
                       {stream.delete[stream];
                        stream ← NIL};
                     RETRY};
              END;
      [] ← NetworkStream.Listen[listenerH: listener];
      stream ← NetworkStream ApproveConnection[
                  listenerH: listener,
                  classOfService: transactional];
      stream.setTimeout[sH: stream, waitTime: 200000];
      Stream SetInputOptions[stream, [terminateOnEndRecord: TRUE]];
      size ← 0; dummy ← 0; why ← sstChange;
        WHILE size = 0
          DO
          IF why = sstChange THEN sst ← dummy;
          [size, why, dummy] ← Stream GetBlock[
                                    stream,
                                    [@b.prefix,0, 2 + BytesInNodeInUse]
                                    ! Stream.TimeOut = > RESUME],
          ENDLOOP;
      IF why # endRecord THEN
          SIGNAL Retry;
      data.nodeNumber ← b.mchnum;
      NodeInUse ← b.usedNode;

      [size, why, sst] ← Stream.GetBlock[
                                  stream,
                                  [@EmptyBlock.prefix,0,BytesInMsgBuffer]
                                  ! Stream.TimeOut = > RESUME];
      IF (size # 0) OR (sst # NetworkStream.closeSST) THEN
          SIGNAL Retry,
```

```
        IF NetworkStream.CloseReply[stream] # good THEN
            SIGNAL Retry;
        stream.delete[stream];
        stream ← NIL;
        DO
          ENABLE BEGIN
                      ABORTED = > EXIT;
                      NetworkStream.ConnectionFailed,
                      NetworkStream.ConnectionSuspended = > {
                      IF stream # NIL THEN
                           {stream.delete[stream];
                            stream ← NIL};
                       WriteString["Receiver – ConnectionFailed or ConnectionSuspended,
                       LOOP"L];
                       LOOP};
                   END;
          [] ← NetworkStream.Listen[listenerH: listener];
          stream ← NetworkStream.ApproveConnection[
                        listenerH: listener,
                        classOfService: transactional],
          Process.Detach[FORK ReceiveInstance[stream]];
        ENDLOOP;
      END;
      NetworkStream.DeleteListener[listener !ABORTED = > CONTINUE];
    END;

  ReceiveInstance: PROCEDURE [stream: Stream.Handle] =
      BEGIN
        tmpPtr: MsgBufferPtr ← NIL;
        EmptyBlock: MACHINE DEPENDENT RECORD[
              prefix: PACKED ARRAY [0..0) OF Stream.Byte,
              pseudo: MsgBuffer];
        size: CARDINAL ← 0;
        why: Stream.CompletionCode ← normal;
        sst, dummy: Stream.SubSequenceType ← 0,

        BEGIN
          ENABLE BEGIN
                      NetworkStream.ConnectionFailed,
                      NetworkStream.ConnectionSuspended = > {
                            IF stream # NIL THEN
                               {stream.delete[stream];
                                stream ← NIL};
                            WriteString["Receiver1 – ConnectionFailed or ConnectionSuspended,
                            CONTINUE"L];
                            CONTINUE};
                    END;
          stream setTimeout[sH: stream, waitTime  200000];
          Stream.SetInputOptions[stream, [terminateOnEndRecord: TRUE]];
          tmpPtr ← zone.NEW[MsgBuffer],
          size ← 0; dummy ← 0; why ← sstChange;
          WHILE size = 0
            DO
            IF why = sstChange THEN sst ← dummy,
            [size, why, dummy] ← Stream.GetBlock[
                                          stream,
                                          [@tmpPtr ↑ .prefix,0,BytesInMsgBuffer]
                                           ! Stream.TimeOut = > {
                                           WriteString["receiver stream timeout"L];
                                           NewLine[];
                                           RESUME}]
            ENDLOOP;
          IF (why # endRecord) OR (size < BytesInMsgBuffer) THEN
              {zone.FREE[@tmpPtr];
               GO TO Exit};
          StoreInData[tmpPtr];

          ]size, why, sst] ← Stream.GetBlock[
                                        stream,
                                        [@EmptyBlock.prefix,0,BytesInMsgBuffer]
                                        ! Stream.TimeOut = > RESUME];
          IF (size # 0) OR (sst # NetworkStream.closeSST) THEN
              GO TO Exit;
          IF NetworkStream.CloseReply[stream] # good THEN
              GO TO Exit;

        EXITS Exit = > NULL;
      END;
      IF stream # NIL THEN
          {stream.delete[stream];
           stream ← NIL};
    END;


  – – main
  Exec.AddCommand[name: "DIBNode.~",
                  proc: Null, help: Help, unload: Unload];
  WindowInit[];
  Process.EnableAborts[@dataAvailable];
  AllNodeList[1].name ← "ALVIS";
  AllNodeList[1].addr ←
          StrIngToNetworkAddress["5772.25200121351.71525",NIL,NIL].addr;
  AllNodeList[2].name ← "LIT";
  AllNodeList[2].addr ←
          StringToNetworkAddress["5772.25200121602.16756",NIL,NIL].addr;
  AllNodeList[3].name ← "FJALAR";
  AllNodeList[3].addr ←
          StringToNetworkAddress["5772.25200121213.13160",NIL,NIL].addr;
  AllNodeList[4].name ← "ANDVARI";
  AllNodeList[4].addr ←
          StringToNetworkAddress["5771.25200122706.22011",NIL,NIL].addr;
  AllNodeList[5].name ← "GALAR";
  AllNodeList[5].addr ←
          StringToNetworkAddress["5772 25200121527.120223",NIL,NIL].addr;
  AllNodeList[6].name ← "BROKK";
  AllNodeList[6].addr ←
          StringToNetworkAddress["5771 25200115272.26065",NIL,NIL].addr;
  AllNodeList[7].name ← "MODSOGNIR";
  AllNodeList[7].addr ←
          StringToNetworkAddress["5771.25200120764.65150",NIL,NIL].addr;
  AllNodeList[B].name ← "DURIN";
  AllNodeList[B].addr ←
          StringToNetworkAddress["5771.25200113474.50653",NIL,NIL].addr;
  AllNodeList[9].name ← "EITRI";
  AllNodeList[9].addr ←
          StringToNetworkAddress["5771.25200123657.35367",NIL,NIL].addr;

  END.
```

```
DIRECTORY
   IO,
   InterfaceDfs,
   NInterfaceDfs,
   Real,
   TTY USING [Handle, SetEcho, GetChar, GetLine, LineOverflow, Rubout, GetDecimal,
    PutChar, PutString, PutDecimal, PutLongDecimal,PutCR],
   TTYSW USING[GetTTYHandle];

NIOImpl: PROGRAM
     IMPORTS TTY,TTYSW,NInterfaceDfs,Real
     EXPORTS IO =

   BEGIN
      data: InterfaceDfs.DataHandle ← NIL;
      ttyWindow: TTY.Handle;

      SetEcho: PUBLIC PROCEDURE [new: BOOLEAN] RETURNS [old: BOOLEAN] =
         BEGIN
            old ← SELECT TTY.SetEcho[h: ttyWindow, new: IF new THEN plain ELSE none]
                  FROM
                     none = > FALSE,
                     plain = > TRUE,
                     ENDCASE = > ERROR
         END «SetEcho»;

      ReadChar: PUBLIC PROCEDURE [] RETURNS [CHARACTER] =
                        {RETURN[TTY.GetChar[ttyWindow]]};

      ReadLine: PUBLIC PROCEDURE [s: LONG STRING] =
         BEGIN
               TTY.GetLine[ttyWindow,s !
                                                   TTY.LineOverflow = >
                                                      {RESUME[SIGNAL LineOverflow[s]]};
                                                   TTY Rubout = > {      SIGNAL Rubout[];
                                                                         RESUME
                                                                  }
                           ]
         END;

      Rubout: PUBLIC SIGNAL [] = CODE;


      LineOverflow: PUBLIC SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING] = CODE,

      ReadDecimal: PUBLIC PROCEDURE [] RETURNS [INTEGER] =
         BEGIN
            RETURN[TTY.GetDecimal[ttyWindow !
                                                   TTY.LineOverflow = >
                                                      {RESUME[SIGNAL LineOverflow[s]]};
                                                   TTY.Rubout = > {      SIGNAL Rubout[];
                                                                         RESUME
                                                                  }
                                    ]]
         END «ReadDecimal»;

      WriteChar: PUBLIC PROCEDURE [c  CHARACTER] = {TTY.PutChar[ttyWindow,c]};

      WriteString: PUBLIC PROCEDURE [s: LONG STRING] =
      {TTY.PutString[ttyWindow,s]};

      WriteDecimal: PUBLIC PROCEDURE [n:INTEGER] =
                              {TTY.PutDecimal[ttyWindow,n]};

      WriteLongDecimal:PUBLIC PROCEDURE [n.LONG INTEGER] =
                                {TTY.PutLongDecimal[ttyWindow,n]};

      WriteFloat: PUBLIC PROCEDURE[n: REAL] =
                {Real.WriteReal[WriteChar,
                                          n,Real.DefaultSinglePrecision ,FALSE]};

      NewLine: PUBLIC PROCEDURE [] = {TTY.PutCR[ttyWindow]};


   – – main
   data ← NInterfaceDfs.GetWindowHandle[];
   ttyWindow ← TTYSW.GetTTYHandle[data.ttySW];

END «IOImpl».
```

```
DIRECTORY
  DeweyDfs USING[Dewey],
  ApplicDfs USING[InfoType, ProblemType],
  InterfaceDfs;

NodeDfs: DEFINITIONS =

  BEGIN
    Open: INTEGER = – 1; – – value for the Worker field in WorkGiven
    < < WorkGiven records all problems we have given to peers to work on.
        It also includes the current problem that we have given to
        ourself to work on, and any problems that are in our heap.
        WorkGotten records all problems that we have received from others,
        including the host. > >
    WorkGottenPtr: TYPE = LONG POINTER TO WorkGottenType;
    WorkGottenType: TYPE = RECORD[
      ID: DeweyDfs.Dewey,
      Owner: INTEGER, – – who is waiting for the results
      UnsolvedCount: INTEGER, < < how many subproblems are still under
                                  computation. > >
      Value: ApplicDfs.ProblemType,
      Next: WorkGottenPtr];
    WorkGivenPtr: TYPE = LONG POINTER TO WorkGivenType;
    WorkGivenType: TYPE = RECORD[
      ID: DeweyDfs.Dewey,
      Value: ApplicDfs.ProblemType,
      Worker: INTEGER, – – who is working on this problem for us
      Available: BOOLEAN, < < may we work on it?
                 If not given to anyone, Worker = – 1 (Open), Available = TRUE
                 If given to machine m, Worker = m, Available = FALSE
                 If given but ready to redo, Worker = m, Available = TRUE
                 If given and I am redoing, Worker = m, Available = FALSE > >
      Parent: WorkGottenPtr,
      Redundancy: INTEGER,
      Next: WorkGivenPtr];

    BroadcastInfo: PROCEDURE[Info: ApplicDfs.InfoType];
    ReportResult: PROCEDURE[P: ApplicDfs.ProblemType];
  END.
```

```
DIRECTORY
  IO,
  DeweyDfs,
  HostOfs,
  InterfaceDfs,
  NInterfaceDfs,
  ApplicDfs,
  NodeDfs,
  TimeMeasureDfs,
  Heap USING[system2one],
  RandomDefs,
  Process;

NodeImpl: PROGRAM
  IMPORTS DeweyDfs,NInterfaceDfs,ApplicDfs,IO,Heap,RandomDefs,
          TimeMeasureDfs,Process
  EXPORTS NodeDfs =

  BEGIN
    OPEN DeweyDfs,HostDfs,NodeDfs,InterfaceDfs,NInterfaceDfs,IO,
         ApplicDfs,RandomDefs,TimeMeasureDfs;

    zone: UNCOUNTED ZONE = Heap.systemZone,
    data: DataHandle ← NIL;

    MaxDeweyReport: INTEGER = 4;
    MaxPartitionSize: INTEGER = 7,
    MaxTimingDepth: INTEGER = 15;
    HelperListType: TYPE = ARRAY[0..MaxPartitionSize – 1] OF INTEGER,
    MyName: INTEGER ← 0;
    Termination: BOOLEAN ← FALSE;
    NextToAsk: INTEGER ← 1;
    HelperList: HelperListType ← ALL[0];
    pgen: LONG POINTER TO GeneratorState ← NIL; – – for random variable

    StartTime: TimeMeasure ← [[2114294400],[1]];
    AvailWorkCount: INTEGER ← 0;
    WorkGivenSize. INTEGER ← 0;
    Debug: INTEGER ← 4; – – debugging level; comes from host
    WorkGivenHeader: WorkGivenPtr ← NIL;
    WorkGottenHeader: WorkGottenPtr ← NIL;

    – – communication variable

    outMsgPtr: MsgBufferPtr ← NIL;
    PartitionSize: INTEGER ← 0;
    NumHelpers: INTEGER ← 0;
    LocalSequence: INTEGER ← 0;
    HaveBeenIdle: BOOLEAN ← TRUE;
    AnswerSpec: AnswerSpecification ← Count;
    DeweyPrefix: Dewey ← [ALL[0],0];
    WorkFract: INTEGER ← 0;

    – – Timing variables.
    Time. TimeType ← ALL[0];
    TimeTemp: INTEGER;
    OldTime1: Millisecs;
    TimingStack: ARRAY[0..MaxTimingDepth] OF INTEGER,
    TimingStackTop: INTEGER;
    Timespent: Millisecs, – – Elapsed time when debug is called

    – – Recursion stack for Backtrack

    CurrentProblemType: TYPE = ARRAY[0..MaxDepth] OF ProblemType;
    Depth: INTEGER ← 0;
    CurrentProblem: LONG POINTER TO CurrentProblemType ← NIL;
    OeweyDelta: ARRAY[1..MaxDepth] OF INTEGER ← ALL[0];
    GiveAwayDepth INTEGER ← 0;
    CurrentRedundancy. INTEGER ← 0;

    Error: PROCEDURE[Condition: BOOLEAN, Reason: LONG STRING] =
      BEGIN
        IF Condition THEN
          {WriteString[" ** error **"L],
          WriteString[Reason];
          NewLine[];
          WHILE TRUE
            DO
            Process.Pause[500];
            ENDLOOP};
      END;

    StartTiming: PROCEDURE[What: INTEGER] =
      BEGIN
        TimingStackTop ← 0;
        TimingStack[TimingStackTop] ← What;
        OldTime1 ← ElapsedTime[StartTime];
      END; – – StartTiming

    PushTiming: PROCEDURE[What: INTEGER] =
      BEGIN
        OldTime1 ← ElapsedTime[StartTime] – OldTime1;
        TimeTemp ← TimingStack[TimingStackTop];
        IF TimeTemp < = 3 + MaxRedundancy THEN
          Time[TimeTemp] ← Time[TimeTemp] + OldTime1;
        TimingStackTop ← TimingStackTop + 1
        TimingStack[TimingStackTop] ← What,
        OldTime1 ← ElapsedTime[StartTime];
      END; – – PushTiming

    PopTiming: PROCEDURE[] =
      BEGIN
        OldTime1 ← ElapsedTime[StartTime] – OldTime1;
        TimeTemp ← TimingStack[TimingStackTop];
        IF TimeTemp < = 3 + MaxRedundancy THEN
          Time[TimeTemp] ← Time[TimeTemp] + OldTime1;
        TimingStackTop ← TimingStackTop – 1;
        Error[TimingStackTop < 0, "Stack error"];
        OldTime1 ← ElapsedTime[StartTime];
      END; – – PopTiming

    DEBUG: PROCEDURE[a: INTEGER, Mess: LONG STRING] =
      BEGIN
        IF Debug > = a THEN
          {PushTiming[TossTime];
           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
           WriteString[" "L];
           WriteString[Mess];
           NewLine[];
           PopTiming[]};
      END; – – DEBUG

    CheckWorkEmpty: PROCEDURE[] =
    < < see if any work left over at end > >
      BEGIN
        Ptr1: WorkGivenPtr←NIL;
        Ptr2: WorkGottenPtr←NIL;

        IF (WorkGottenHeader # NIL) OR
            (WorkGivenHeader # NIL) THEN
            {DEBUG[3,"Entries in WorkGiven. "],
             Ptr1 ← WorkGivenHeader;
             WHILE Ptr1 # NIL
               DO
               IF Debug > = 2 THEN
                 {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
                  WriteString[" "L];
                  PrintDewey[Ptr1 ↑ .ID];
                  NewLine[]};
               Ptr1← Ptr1 ↑ .Next;
               ENDLOOP;
             DEBUG[3,"Entries in WorkGotten: "],
             Ptr2 ← WorkGottenHeader,
             WHILE Ptr2 # NIL
               DO
               IF Debug > = 3 THEN
                 {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                  WriteString[" "L],
```

```
                PrintDewey[Ptr2↑.ID],
                NewLine[];
              Ptr2← Ptr2↑.Next;
              ENDLOOP};
    END; - - CheckWorkEmpty

Giving: PROCEDURE[ParPtr: WorkGottenPtr,
                  Prob: ProblemType,
                  Dew: Dewey,
                  Who: INTEGER,
                  Redun: INTEGER] =
<< We are about to give 'Prob' to 'Who'.  Record this fact in
   WorkGiven. >>
  BEGIN
    TmpPtr: WorkGivenPtr ← NIL;

    IF Debug > = 2 THEN
      {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
       WriteString[" Giving "L];
       PrintDewey[Dew];
       WriteString[" > "L[;
       WriteDecimal[Who[;
       NewLine[]};
    IF WorkGivenHeader = NIL THEN
      {WorkGivenHeader ← zone.NEW[WorkGivenType];
       WorkGivenHeader↑.Next ← NIL}
    ELSE {TmpPtr ← WorkGivenHeader;
          WorkGivenHeader ← zone.NEW[WorkGivenType[;
          WorkGivenHeader↑.Next ← TmpPtr},
    {OPEN WorkGivenHeader↑;
     ID← Dew;
     Value← Prob;
     Worker← Who;
     Error[Who = 0, "Bad Who"[;
     Available← (Worker = Open);
     IF Available THEN
       AvailWorkCount ← AvailWorkCount + 1
     Redundancy← Redun;
     Error[Redundancy < 0, "Bad Redundancy"[;
     Parent← ParPtr;
     Parent↑.UnsolvedCount ← Parent↑.UnsolvedCount + 1};
    WorkGivenSize← WorkGivenSize + 1;
  END; - - Giving

Getting: PROCEDURE[Dew: Dewey,Prob: ProblemType,Who: INTEGER]
                  RETURNS[ParPtr: WorkGottenPtr] =
<< We have just gotten 'Dew' from 'Who'.  Record this fact in
   Workgotten. >>
  BEGIN
    TmpPtr: WorkGottenPtr;

    IF Debug > = 2 THEN
      {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
       WriteString[" Getting "L],
       PrintDewey[Dew];
       WriteString[" < "L];
       WriteDecimal[Who[;
       NewLine[]};
    IF WorkGottenHeader = NIL THEN
      {WorkGottenHeader ← zone.NEW[WorkGottenType];
       WorkGottenHeader↑.Next ← NIL}
    ELSE {TmpPtr ← WorkGottenHeader,
          WorkGottenHeader ← zone.NEW[WorkGottenType];
          WorkGottenHeader↑.Next ← TmpPtr};
    ParPtr← WorkGottenHeader;
    {OPEN ParPtr↑;
     ID← Dew;
     Owner← Who,
     Value← Prob;
     UnsolvedCount← 0};
  END; - - Getting

  FindWorkGotten: PRDCEDURE[Dew. Dewey]
                  RETURNS[Answer. WorkGottenPtr] =
<< Someone has just solved a problem.  The dewey prefix is given
   by Dew.  Find which entry in WorkGotten applies.  If none does,
   return null; that means that the problem has been solved
   redundantly. >>
  BEGIN
    Answer ← WorkGottenHeader;
    WHILE DeweyCompare[Answer↑.ID, Dew] # Equal
      DO
      Answer ← Answer↑.Next;
      IF Answer = NIL THEN
        {IF Debug > = 3 THEN
           {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
            WriteString[" "L];
            WriteString["No such WorkGotten: "L];
            PrintDewey[Dew[;
            NewLine[]};
         EXIT};
      ENDLOOP;
  END; - - FindWorkGotten

FindWorkGiven: PROCEDURE[Dew. Dewey]
                  RETURNS[Answer. WorkGivenPtr] =
<< We have just heard from a beggar the results of its work that we
   originated. Return Who the parent is. If there is no correspoding
   WorkGiven (someone else has already finished this work or we are
   called an extra time), then return nil. >>
  BEGIN
    Answer ← WorkGivenHeader;
    WHILE Answer # NIL
      DO
      IF DeweyCompare[Answer↑.ID, Dew] = Equal THEN
        EXIT;
      Answer ← Answer↑.Next;
      ENDLOOP,
  END; - - FindWorkGiven

ReleaseWorkGiven. PROCEDURE[Which: WorkGivenPtr] =
<< 'Which' points to some work that we gave away that is now finished.
   It will not be nil; if someone else finished the work, then we are
   not called. Remove it from the WorkGiven structure. There will not
   be any other redundant copies of the same problem (as seen from
   the dewey number. >>
  BEGIN
    Fore: WorkGivenPtr;
    Aft: WorkGivenPtr;

    Fore ← WorkGivenHeader;
    WHILE Fore # Which
      DO
      Aft ← Fore;
      Fore ← Fore↑.Next;
      ENDLOOP;
    IF Debug > = 2 THEN
      {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
       WriteString[" Removing ["L];
       WriteDecimal[Which↑.Redundancy];
       WriteChar[']],
       PrintDewey[Which↑.ID];
       NewLine[]},
    WorkGivenSize ← WorkGivenSize - 1;
    IF Which↑.Available THEN
      AvailWorkCount ← AvailWorkCount - 1,
    IF Fore = WorkGivenHeader THEN
      WorkGivenHeader ← WorkGivenHeader↑.Next
    ELSE Aft↑.Next ← Fore↑.Next;
    zone.FREE[@Which];
  END, - - ReleaseWorkGiven

ReleaseSubsumedWork: PROCEDURE[Dew: Dewey] =
  BEGIN
    Fore, Aft: WorkGivenPtr ← NIL;

    WHILE DeweySubsumes]Dew,WorkGivenHeader↑.ID]
      DO
```

```
        Aft ← WorkGivenHeader;
        WorkGivenHeader ← WorkGivenHeader↑.Next;
        zone.FREE[@Aft];
        ENDLOOP;
      Aft ← WorkGivenHeader;
      Fore ← Aft↑.Next;
      WHILE Fore # NIL
        DO
        IF DeweySubsumes[Dew, Fore↑.ID] THEN
          {IF Debug > = 2 THEN
              {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                WriteString[" Excise ["L];
                WriteDecimal[Fore↑.Redundancy];
                WriteChar[']];
                PrintDewey[Fore↑.ID];
                NewLine[]};
            WorkGivenSize ← WorkGivenSize – 1;
            IF Fore↑.Available THEN
              AvailWorkCount ← AvailWorkCount – 1;
            Aft↑.Next ← Fore↑.Next;
            zone.FREE[@Fore]}
          ELSE Aft ← Aft.Next;
          Fore ← Aft↑.Next;
          ENDLOOP;
        END; – – ReleaseSubsumedWork

  ReleaseWorkGotten: PROCEDURE[Which: WorkGottenPtr] =
  BEGIN
    Fore,Aft: WorkGottenPtr;

    Fore ← WorkGottenHeader;
    WHILE Fore # Which
      DO
      Aft ← Fore;
      Fore ← Fore↑.Next;
      ENDLOOP;
    IF Fore = WorkGottenHeader THEN
      WorkGottenHeader ← WorkGottenHeader↑.Next
    ELSE Aft↑.Next ← Fore↑.Next,
    zone.FREE[@Which];
  END; – – ReleaseWorkGotten

WorkClear: PROCEDURE[Redun: INTEGER] =
< < If we have any work in WorkGiven that has greater redundany than
   Redun, reset its redundancy level to Redun, because that is the
   level we are noe returning to. > >
  BEGIN
    Fore: WorkGivenPtr;

    Fore ← WorkGivenHeader;
    WHILE Fore # NIL
      DO
      IF Fore↑.Redundancy > Redun THEN
        Fore↑.Redundancy ← Redun;
      Fore ← Fore↑.Next;
    ENDLOOP;
  END; – – WorkClear

ReDoing. PROCEDURE[] RETURNS[Work. WorkGivenPtr] =
< < pick some work we have given away and mark it as Available. Return
its information (Work) and increment its redundancy. > >
  BEGIN
    NewWork: WorkGivenPtr;
    TmpPtr: WorkGivenPtr;

    Error[WorkGivenHeader = NIL, "Nothing on WorkGiven "];
    NewWork ← WorkGivenHeader;
    TmpPtr ← NewWork↑.Next;
    WHILE TmpPtr # NIL
      DO
      IF (TmpPtr↑.Redundancy < NewWork↑.Redundancy) OR
          ((TmpPtr↑.Redundancy = NewWork↑.Redundancy) AND
          (DeweyCompare[TmpPtr↑.ID,NewWork↑.ID] = Smaller)) THEN
          NewWork ← TmpPtr;
        TmpPtr ← TmpPtr↑.Next;
        ENDLOOP;
      Error[NewWork↑.Worker = 0, "Bad Worker "];
      Error[NewWork↑.Worker < 0, "Open Worker "];
      NewWork↑.Redundancy ← NewWork↑.Redundancy + 1;
      Work ← NewWork;
      NewWork↑.Available ← TRUE;
      AvailWorkCount ← AvailWorkCount + 1;
  END; – – ReDoing

SelectWork: PROCEDURE[MyName: INTEGER] RETURNS[SomeWork: WorkGivenPtr] =
< < Return in SomeWork the best open problem, marking it as no longer
   open. Also decrement AvailWorkCount in the process. Select is
   never called if there are no open problems, so there is no worry
   about falling off the end of the WorkGiven list. > >
  BEGIN
    Ptr: WorkGivenPtr;

    SomeWork ← WorkGivenHeader,
    WHILE NOT SomeWork↑.Available
      DO
      SomeWork ← SomeWork↑.Next;
      ENDLOOP,
    Ptr ← SomeWork↑.Next,
    WHILE Ptr # NIL
      DO
      IF Ptr↑.Available AND
        (DeweyCompare[Ptr↑.ID, SomeWork↑.ID] = Smaller) THEN
        SomeWork ← Ptr;
      Ptr ← Ptr↑.Next;
      ENDLOOP;
    {OPEN SomeWork↑,
     IF Worker = Open THEN
       Worker ← MyName,
     Available ← FALSE;
     AvailWorkCount ← AvailWorkCount – 1,
     IF Debug > = 3 THEN
       {WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
         WriteString[" Select "L],
         PrintDewey[SomeWork↑.ID];
         PrintProblem[Value];
         NewLine[]}};
  END, – – SelectWork

HouseKeeping: PROCEDURE[] =
  BEGIN
    next1. WorkGivenPtr ← NIL,
    next2: WorkGottenPtr ← NIL;

    WHILE WorkGivenHeader # NIL
      DO
      next1 ← WorkGivenHeader;
      WorkGivenHeader ← WorkGivenHeader↑.Next;
      zone.FREE[@next1];
      ENDLOOP,
    WHILE WorkGottenHeader # NIL
      DO
      next2 ← WorkGottenHeader,
      WorkGottenHeader ← WorkGottenHeader↑.Next;
      zone.FREE[@next2];
      ENDLOOP;
    WorkGivenSize ← 0;
    AvailWorkCount ← 0;
    HaveBeenIdle ← TRUE;
    ClearInQueue[];
  END; – – WorkInit

  BroadcastInfo: PUBLIC PROCEDURE[Info: InfoType] =
  < < send it to everyone. > >
    BEGIN
      ToWhom: INTEGER ← 1;
      a: InterNodeMessage Message;

      PushTiming[CommTime];
```

```
    WHILE ToWhom < = PartitionSize
      DO
        IF ToWhom # MyName THEN
          {outMsgPtr ← zone.NEW[MsgBuffer];
           a.Kind ← GlobalInfo;
           a.SomeInfo ← Info;
           outMsgPtr ↑ .message ← a;
           outMsgPtr ↑ .source ← MyName;
           outMsgPtr ↑ .dest ← ToWhom;
           Sender[outMsgPtr]};
        ToWhom ← ToWhom + 1;
      ENDLOOP;
    PopTiming[];
  END; – – TellInfo

ReportResult: PUBLIC PROCEDURE[P: ProblemType] =
< < send a result value back to the host > >
  BEGIN
    a: AnswerMessage Message;

    IF AnswerSpec = Full THEN
      {PushTiming[TossTime];
       outMsgPtr ← zone.NEW[MsgBuffer];
       a.Status ← AnAnswer;
       a.Prob ← P;
       a.Sequence ← LocalSequence;
       LocalSequence ← LocalSequence + 1
       outMsgPtr ↑ .message ← a;
       outMsgPtr ↑ .source ← MyName;
       outMsgPtr ↑ .dest ← 0;
       Sender[outMsgPtr];
       PopTiming[]};
  END; – – ReportResult

DropOut: PROCEDURE[] =
< < tell host we are quitting. > >
  BEGIN
    Seq: INTEGER ← 0;
    a: AnswerMessage Message;

    PushTiming[TossTime];
    outMsgPtr ← zone.NEW[MsgBuffer];
    a.Status ← Quitting;
    a.Sequence ← LocalSequence;
    a.Times ← Time;
    LocalSequence ← LocalSequence + 1;
    outMsgPtr ↑ .message ← a;
    outMsgPtr ↑ .source ← MyName;
    outMsgPtr ↑ .dest ← 0;
    Sender[outMsgPtr];
    CheckWorkEmpty[];
    IF MyName = 1 THEN
      {outMsgPtr ← zone.NEW[MsgBuffer];
       a.Status ← PleaseStop;
       a.Sequence ← LocalSequence;
       LocalSequence ← LocalSequence + 1;
       outMsgPtr ↑ .message ← a;
       outMsgPtr ↑ .source ← MyName;
       outMsgPtr ↑ .dest ← 0;
       Process.Pause[60];
       Sender[outMsgPtr]};
    PopTiming[];
  END; – – DropOut

TellTerminate: PROCEDURE[] =
< < only called in node 1. Sends a 'terminate' message to every other
   node and sets the 'termination' flag in node 1 > >
  BEGIN
    Node. INTEGER ← 2;
    a: InterNodeMessage Message,

    PushTiming[CommTime];
    DEBUG[2, "In TellTerminate "];
    WHILE Node < = PartitionSize
      DO
        outMsgPtr ← zone.NEW[MsgBuffer];
        a.Kind ← Terminate;
        outMsgPtr ↑ .message ← a;
        outMsgPtr ↑ .source ← MyName;
        outMsgPtr ↑ .dest ← Node;
        Sender[outMsgPtr];
        Node ← Node + 1;
      ENDLOOP;
    Termination ← TRUE;
    PopTiming[];
  END; – – TellTerminate

BugHost: PROCEDURE[ID: Dewey] =
< < We have finished a subproblem (or the whole problem); tell the
   Dewey number of what we just finished to the host. > >
  BEGIN
    a: AnswerMessage Message;

    PushTiming[TossTime];
    outMsgPtr ← zone.NEW[MsgBuffer];
    a.Status ← Reporting;
    a.AnswerDewey ← ID;
    a.Sequence ← LocalSequence;
    LocalSequence ← LocalSequence + 1;
    outMsgPtr ↑ .message ← a;
    outMsgPtr ↑ .source ← MyName;
    outMsgPtr ↑ .dest ← 0;
    Sender[outMsgPtr];
    PopTiming[];
  END; – – BugHost

TellHost: PROCEDURE[Root: ProblemType] =
< < We have finished the whole problem; tell the updated root to
   the host. > >
  BEGIN
    a: AnswerMessage Message;

    PushTiming[TossTime];
    outMsgPtr ← zone.NEW[MsgBuffer],
    a.Status ← BatchDone;
    a.Prob ← Root;
    a.Times ← Time;
    a.Sequence ← LocalSequence;
    LocalSequence ← LocalSequence + 1;
    outMsgPtr ↑ .message ← a;
    outMsgPtr ↑ .source ← MyName;
    outMsgPtr ↑ dest ← 0;
    Sender[outMsgPtr];
    PopTiming[];
  END; – – TellHost

WorkDone: PROCEDURE[Dew: Dewey,
                    ProbPtr: LONG POINTER TO ProblemType] =
< < Either we or someone else has finished this problem; tell each
   parent There may be several parents, since we may have the same
   problem at different redundancies. > >
  BEGIN
    GivenWork. WorkGivenPtr,
    Parent: WorkGottenPtr;

    DO
      GivenWork ← FindWorkGiven[Dew];
      IF GivenWork = NIL THEN EXIT;
      Parent ← GivenWork ↑ Parent;
      ReleaseWorkGiven[GivenWork];
      TellParent[ProbPtr,Dew,Parent];
    ENDLOOP;
  END; – – WorkDone

TellRepeating: PROCEDURE[GivenWork: WorkGivenPtr] =
< < I have decided to do GivenWork again. Tell its owner,
   (fault – tolerance rule a) so that if the owner has other work for
   me to do, it will tell me. > >
```

```
    BEGIN
      a: InterNodeMessage Message;

      PushTiming[CommTime];
      {OPEN GivenWork↑;
        outMsgPtr ← zone.NEW[MsgBuffer];
        a.Kind ← Repeating;
        a.Asker ← MyName;
        a.Redundancy ← Redundancy;
        a.ChildDewey ← ID;
        outMsgPtr↑.message ← a;
       outMsgPtr↑.source ← MyName;
       outMsgPtr↑.dest ← Worker};
       Sender[outMsgPtr];
       PopTiming[];
    END; – – TellRepeating

  TellUpdate: PROCEDURE[GivenPtr: WorkGivenPtr] =
  << The problem in GivenPtr↑.Value has changed, so we must tell the
      worker we have given it to (it's not us, and it's not free)
      about the change. >>
    BEGIN
      a: InterNodeMessage Message;

      PushTiming[CommTime];
      outMsgPtr ← zone.NEW[MsgBuffer];
      a.Kind ← Updating;
      a.Child ← GivenPtr↑.Value;
      a.ChildDewey ← GivenPtr↑.ID;
      IF Debug >= 3 THEN
          {PushTiming[TossTime];
           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
           WriteString["TellUpdate > "L],
           WriteDecimal[GivenPtr↑.Worker];
           PopTiming[]};
      outMsgPtr↑.message ← a;
      outMsgPtr↑.source ← MyName;
      outMsgPtr↑.dest ← GivenPtr↑.Worker;
      Sender[outMsgPtr];
      PopTiming[];
    END; – – TellUpdate

  DoUpdate: PROCEDURE[Dew: Dewey,
                      ProbPtr: LONG POINTER TO ProblemType] =
  << The children of Prob, with ID = Dew, need to have Update called,
      recursively if necessary. >>
    BEGIN
      D: INTEGER;
      GivenPtr: WorkGivenPtr;
      Again: BOOLEAN;

      GivenPtr ← WorkGivenHeader;
      WHILE GivenPtr # NIL
        DO
        {OPEN GivenPtr↑;
         IF (ID.Length = Dew.Length + 1) AND
              DeweySubsumes[Dew, ID] THEN
            SELECT TRUE FROM
              (Worker # MyName) AND (NOT Available) = >
                {Again ← Update[@Value,ProbPtr];
                 IF Again THEN
                    {WriteString["'"L];
                     TellUpdate[GivenPtr]}};
              DeweyCompare[ID,DeweyPrefix] = Equal = >
                {Again ← Update[@CurrentProblem[0],@Value];
                 D ← 1;
                 WHILE Again AND (D < Depth)
                    DO
                    WriteString["'"L];
                    Again ← Update[@CurrentProblem[D],@CurrentProblem[D – 1]],
                    D ← D + 1;
                    ENDLOOP};
              ENDCASE = > {Again ← Update[@Value,ProbPtr];
                           IF Again THEN
                              {WriteString["'"L];
                               DoUpdate[ID, @Value]}}};
        GivenPtr ← GivenPtr↑.Next;
        ENDLOOP;
    END; – – DoUpdate

  TellParent: PROCEDURE[ChildPtr: LONG POINTER TO ProblemType,
                        ChildDewey: Dewey,
                        Parent: WorkGottenPtr] =
  << Child has been finished. Its parent in WorkGotten is pointed to by
      Parent. Update the Parent and inform its owner if all its children
      have finished. >>
    BEGIN
      NeedUpdate: BOOLEAN;
      a: InterNodeMessage Message;

      DEBUG[3, " TellParent "];
      IF DeweyCompare[Parent↑.ID, ChildDewey] # Equal THEN
         {NeedUpdate ← Combine[ChildPtr,@Parent↑.Value];
          IF NeedUpdate THEN
             DoUpdate[Parent↑.ID, @Parent↑.Value]}
      ELSE Parent↑.Value ← ChildPtr↑;
      Parent↑.UnsolvedCount ← Parent↑.UnsolvedCount – 1;
      IF Parent↑.UnsolvedCount = 0 THEN
         {IF Parent↑.Owner = MyName THEN
             WorkDone[Parent↑.ID, @Parent↑.Value]
          ELSE {IF Parent↑.Owner = 0 THEN
                   {IF Debug >= 1 THEN
                       {WriteString["In TellParent, owner is 0... "L],
                        WriteLongDecimal[Parent↑.Value.LeafCount];
                        NewLine[]};
                    TellHost[Parent↑.Value];
                    TellTerminate[]}
                ELSE {
                    PushTiming[CommTime];
                    outMsgPtr ← zone.NEW[MsgBuffer];
                    a.Kind ← Result;
                    a.ChildDewey ← Parent↑.ID;
                    a.Child ← Parent↑.Value;
                    IF Debug >= 2 THEN
                        {PushTiming[TossTime];
                         WriteLongDecimal[Timespent ←
                         ElapsedTime[StartTime]],
                         WriteString["Tell ["L];
                         WriteDecimal[CurrentRedundancy];
                         WriteChar[']];
                         PrintDewey[Parent↑.ID];
                         WriteString[" > "L];
                         WriteDecimal[Parent↑ Owner];
                         PopTiming[]};
                    outMsgPtr↑ message ← a;
                    outMsgPtr↑.source ← MyName;
                    outMsgPtr↑.dest ← Parent↑.Owner;
                    Sender[outMsgPtr];
                    IF Parent↑.ID.Length < MaxDeweyReport THEN
                       BugHost[Parent↑.ID];
                    PopTiming[]}};
          ReleaseWorkGotten[Parent]};
    END; – – TellParent

  AskForHelp: PROCEDURE[FirstTime: BOOLEAN, Beggar, Hops: INTEGER] =
  << If FirstTime then we ask node1; afterwards, send a request to
      NumHelpers nodes next in line If Beggar # MyName, we are forwarding
      the request, so we only send out one copy. Tell the helper the
      number of hops. >>
    BEGIN
      Seq, Index, Count, Temp: INTEGER,
      Helpers: HelperListType;
      a: InterNodeMessage Message;

      PushTiming[CommTime];
      IF FirstTime OR (Beggar # MyName) THEN
         Count ← 1
      ELSE Count ← NumHelpers,
```

```
    IF (Beggar # MyName) AND (PartitionSize = 2) THEN
      Count ← 0;
    < < Seq ← 1;
    WHILE Seq < = Count
      DO
      Helpers[PartitionSize – Seq – 1] ← NextToAsk;
      NextToAsk ← NextToAsk MOD PartitionSize + 1;
      IF NextToAsk = MyName THEN
        NextToAsk ← NextToAsk MOD PartitionSize + 1;
      Seq ← Seq + 1;
      ENDLOOP; > >
    Helpers ← HelperList;
    Seq ← 1;
    WHILE Seq < = Count
      DO
      Index ← Random[pgen,(PartitionSize – Seq)];
      Temp ← Helpers[Index];
      Helpers[Index] ← Helpers[PartitionSize – Seq – 1];
      Helpers[PartitionSize – Seq – 1] ← Temp;
      IF Temp # Beggar THEN
        Seq ← Seq + 1;
      – – ELSE try again
      ENDLOOP;
    IF FirstTime THEN
      Helpers[PartitionSize – 2] ← 1;
    Seq ← 1;
    WHILE Seq < = Count
      DO
      outMsgPtr ← zone.NEW[MsgBuffer];
      a.Kind ← Request;
      a.Asker ← Beggar;
      a.Count ← Hops;
      outMsgPtr ↑ .message ← a,
      IF Debug > = 2 THEN
        {PushTiming[TossTime];
         WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
         WriteString[" Asking "L];
         WriteDecimal[Helpers[PartitionSize – Seq – 1]],
         PopTiming[]};
      outMsgPtr ↑ .source ← MyName;
      outMsgPtr ↑ .dest ← Helpers[PartitionSize – Seq – 1];
      Sender[outMsgPtr];
      Seq ← Seq + 1;
      ENDLOOP;
    PopTiming[];
  END; – – AshForHelp

HandleRequest: PROCEDURE[Who, Hops: INTEGER] =
< < Who has just asked us for work, or we have decided out of our
    nature to foist some work on Who If we have any to spare, send
    it; otherwise, forward the request, if it is forwardable. If it
    has already got Hops = PartitionSize, don't forward it 'Repeating'
    messages don't get forwarded this way. > >
  BEGIN
    P: ProblemType;
    First, Done: BOOLEAN;
    GiveAwayCount: INTEGER;
    TmpDewey: Dewey;
    WorkPtr WorkGivenPtr;
    SomeWork. WorkGivenPtr;
    ParPtr. WorkGottenPtr;
    Seq:INTEGER,
    a. InterNodeMessage Message;

    PushTiming[TimeMeasures + CurrentRedundancy];
    IF Debug > = 2 THEN
      {PushTiming[TossTime];
       WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
       WriteString["Asked by "L];
       WriteDecimal[Who];
       PopTiming[]};
    WHILE AvailWorkCount = 0
      DO
      IF GiveAwayDepth > = Depth THEN
        {IF Debug > = 2 THEN
          {PushTiming[TossTime];
           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
           WriteString["Too deep to give away; depth "L];
           WriteDecimal[GiveAwayDepth];
           PopTiming[]};
         EXIT};
      IF NOT NonTrivial[CurrentProblem[GiveAwayDepth + 1]] THEN
        {IF Debug > = 2 THEN
          {PushTiming[TossTime];
           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
           WriteString["Trivial when deeper than "L];
           WriteDecimal[GiveAwayDepth];
           PopTiming[]},
         EXIT};
      WorkPtr ← FindWorkGiven[DeweyPrefix];
      IF WorkPtr = NIL THEN
        {DEBUG[2, "nothing at the moment"];
         EXIT};
      IF DeweyCompare[WorkPtr ↑ .ID, WorkPtr ↑ .Parent ↑ .ID] = Equal THEN
        {ParPtr ← WorkPtr ↑ .Parent;
         ParPtr ↑ Value ← WorkPtr ↑ .Value;
         ReleaseWorkGiven[WorkPtr];
         ParPtr ↑ UnsolvedCount ← ParPtr ↑ .UnsolvedCount – 1}
      ELSE ParPtr ← Getting[WorkPtr ↑ .ID, WorkPtr ↑ .Value, MyName];
      GiveAwayDepth ← GiveAwayDepth + 1;
      {OPEN DeweyPrefix;
       Length ← Length + 1;
       DNums[Length] ← DeweyDelta[GiveAwayDepth]};
      P ← CurrentProblem[GiveAwayDepth];
      Giving[ParPtr,P,DeweyPrefix,MyName,CurrentRedundancy];
      TmpDewey ← DeweyPrefix;
      First ← FALSE;
      DO
        Done ← Generate[First,@ParPtr ↑ .Value,@P];
        IF Done THEN EXIT;
        {OPEN TmpDewey;
         DNums[Length] ← DNums[Length] + 1};
        Giving[ParPtr,P,TmpDewey,Open,CurrentRedundancy],
      ENDLODP;
      IF AvailWorkCount > 0 THEN EXIT;
    ENDLOOP;
    GiveAwayCount ← AvailWorkCount;
    IF WorkFract < 0 THEN
      {IF GiveAwayCount > 0 THEN
          GiveAwayCount ← 1}
    ELSE {IF GiveAwayCount > 0 THEN
              {GiveAwayCount ← ((GiveAwayCount * WorkFract) + 5) / 10,
               IF GiveAwayCount = 0 THEN
                  GiveAwayCount ← 1}};
    PopTiming[];
    PushTiming[CommTime],
    IF GiveAwayCount > 0 THEN
      {Seq ← 1;
       WHILE Seq < = GiveAwayCount
          DO
          outMsgPtr ← zone.NEW[MsgBuffer],
          SomeWork ← SelectWork[MyName];
          IF Debug > = 2 THEN
              {PushTiming[TossTime],
               WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
               WriteString["Give ["L],
               WriteDecimal[CurrentRedundancy];
               WriteChar[']];
               PrintDewey[SomeWork ↑ ID];
               WriteString[" > "L],
               WriteDecimal[Who];
               PopTiming[]};
          a.Kind ← Work;
          a.Count ← 1;
          a.Child ← SomeWork ↑ Value;
          a.ChildDewey ← SomeWork ↑ ID;
          a.Redundancy ← SomeWork ↑ .Redundancy,
          Error[a Redundancy < 0, "Bad redundancy"];
```

```
            IF DeweyCompare[SomeWork↑.ID, SomeWork↑.Parent↑.ID] = Equal THEN
                {a.Asker ← SomeWork↑.Parent↑.Owner;
                 ReleaseWorkGotten[SomeWork↑.Parent];
                 ReleaseWorkGiven[SomeWork]}
            ELSE {a.Asker ← MyName;
                  SomeWork↑.Worker ← Who;
                  Error[Who = 0, "Bad Who"]};
            outMsgPtr↑.message ← a;
            outMsgPtr↑.source ← MyName;
            outMsgPtr↑.dest ← Who;
            Sender[outMsgPtr];
            Seq ← Seq + 1;
            ENDLOOP}
        ELSE IF Hops < PartitionSize THEN
                AskForHelp[FALSE,Who,Hops + 1],
      PopTiming[];
    END; -- HandleRequest

  CheckIncoming: PROCEDURE[BusyWait: BOOLEAN]
                                RETURNS[Abort: BOOLEAN] =
  << called periodically to see if our predecessor node has asked or
     forwarded to us a request for some work to do. If it has, we will
     give half of our outstanding work to the requester. If we have no
     work to hive away, we will split the current level of
     CurrentProblem if it is not trivial; otherwise we will pass the
     request to our successor. If BusyWait is set, we don't return
     if nothing has come in; we keep checking. We set Abort to true
     if work just came in that is less redundant than the current
     level. >>
  BEGIN
    GivenWork: WorkGivenPtr;
    ParPtr: WorkGottenPtr;
    Empty: BOOLEAN;
    inMsgPtr: MsgBufferPtr ← NIL;
    askAgain: CARDINAL ← 0;

    PushTiming[CommTime];
    Abort ← FALSE;
    DO
      Empty ← InQueueEmptyCheck[],
      IF NOT Empty THEN
        {inMsgPtr ← GetInData[];
         WITH a  inMsgPtr↑.message SELECT FROM
           InterNodeMessage => {
             SELECT TRUE FROM
               (a.Kind = Terminate) OR Termination => Termination ← TRUE;
               a.Kind = Request =>
                 {IF a.Asker # MyName THEN
                     HandleRequest[a.Asker, a.Count];
                  DEBUG[1, " Got request "],
                  IF (TimingStack[TimingStackTop] = IdleTime) AND
                     (NOT HaveBeenIdle) THEN
                     {AskForHelp[FALSE,MyName,PartitionSize];
                      HaveBeenIdle ← TRUE}},
               a.Kind = Updating => DoUpdate[a.ChildDewey, @a.Child];
               a.Kind = Work =>
                 {IF a.Redundancy < CurrentRedundancy THEN
                     {WorkClear[a.Redundancy];
                      Error[a.Redundancy < 0, "Got bad redundancy"],
                      Abort ← TRUE;
                      CurrentRedundancy ← a.Redundancy};
                  IF DeweySubsumes[DeweyPrefix, a.ChildDewey] THEN
                     {IF Debug >= 1 THEN
                         {PushTiming[TossTime];
                          WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
                          WriteString[" #c#: got "L];
                          PrintDewey[a.ChildDewey];
                          WriteString[" while working on "L],
                          PrintDewey[DeweyPrefix],
                          PopTiming[]},
                      Abort ← TRUE,
                      CurrentRedundancy ← a.Redundancy},
                  IF a.Redundancy = CurrentRedundancy THEN
                     {IF FindWorkGiven[a.ChildDewey] # NIL AND Debug >= 3 THEN
                         {PushTiming[TossTime];
                          WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                          WriteString[" Getting similar work: "L];
                          PrintDewey[a.ChildDewey];
                          PopTiming[]};
                      ParPtr ← Getting[a.ChildDewey,a.Child,a.Asker];
                      Giving[ParPtr,a.Child,a.ChildDewey,Open,CurrentRedundancy];
                      BusyWait ← FALSE;
                      IF Debug >= 3 THEN
                         {PushTiming[TossTime];
                          WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                          WriteString["Get ["L],
                          WriteDecimal[a.Redundancy];
                          WriteChar[']];
                          PrintDewey[a.ChildDewey];
                          WriteString[" <- "L];
                          WriteDecimal[inMsgPtr↑.source];
                          PopTiming[]};
                      GivenWork ← WorkGivenHeader;
                      WHILE GivenWork # NIL
                         DO
                         IF (NOT GivenWork↑.Available) AND
                            (GivenWork↑.Worker # MyName) AND
                            DeweySubsumes[a.ChildDewey,GivenWork↑.ID] THEN
                            {IF Debug >= 1 THEN
                                {PushTiming[TossTime];
                                 WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                                 WriteString[" #c#: reprieving "L];
                                 WriteDecimal[GivenWork↑.Worker];
                                 NewLine[];
                                 PopTiming[]};
                             HandleRequest[GivenWork↑.Worker, PartitionSize]};
                         GivenWork ← GivenWork↑.Next;
                         ENDLOOP}
                  ELSE {IF Debug >= 3 THEN
                          {PushTiming[TossTime];
                           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                           WriteString["Ignoring ["L];
                           WriteDecimal[a.Redundancy];
                           WriteString["] "L],
                           PrintDewey[a.ChildDewey];
                           WriteString[" < "L];
                           WriteDecimal[inMsgPtr↑.source];
                           PopTiming[]}}};
               a.Kind = Result =>
                 {IF Debug >= 2 THEN
                     {PushTiming[TossTime];
                      WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
                      WriteString["Report "L];
                      PrintDewey[a.ChildDewey];
                      WriteString[" < "L];
                      WriteDecimal[inMsgPtr↑.source];
                      PopTiming[]};
                  PushTiming[TimeMeasures + CurrentRedundancy];
                  WorkDone[a.ChildDewey, @a.Child];
                  IF DeweySubsumes[a.ChildDewey,DeweyPrefix] THEN
                     {DEBUG[2, " stopped redundant work "];
                      Abort ← TRUE};
                  PopTiming[]};
               a.Kind = GlobalInfo =>
                 {PushTiming[TimeMeasures + CurrentRedundancy],
                  UseNewInfo[a.SomeInfo];
                  PopTiming[]},
               a.Kind = Repeating =>
                 {IF Debug >= 1 THEN
                     {PushTiming[TossTime];
                      WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                      WriteString[" #b#: "L],
                      WriteDecimal[inMsgPtr↑.source];
                      WriteString[" is repeating "L];
                      PrintDewey[a.ChildDewey];
                      PopTiming[]};
                  IF a.Redundancy >= CurrentRedundancy THEN
                     {DEBUG[1, " we will treat as a request"],
```

```
                    HandleRequest[a.Asker, PartitionSize]}};
              ENDCASE = > Error[TRUE, "Bad message type"];
              zone.FREE[@inMsgPtr]};
            ENDCASE}
          ELSE {IF Termination OR Abort OR (NOT BusyWait) THEN
                  EXIT
                ELSE {IF WorkGivenSize > 0 THEN
                        {GivenWork ← ReDoing[];
                         Error[GivenWork ↑ .Worker < = 0, "Bad Worker"];
                         TellRepeating[GivenWork];
                         IF Debug > = 1 THEN
                          {PushTiming[TossTime];
                           WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                           WriteString[" #a#: Redo ["L];
                           WriteDecimal[GivenWork ↑ .Redundancy];
                           WriteString["] < "L];
                           WriteDecimal[GivenWork ↑ .Worker];
                           PrintDewey[GivenWork ↑ .ID];
                           PopTiming[]};
                         EXIT}}};
      TimingStack[TimingStackTop] ← IdleTime;
      askAgain ← askAgain + 1;
      IF askAgain > 60000 THEN
          {askAgain ← 0;
           DEBUG[2, "Asking again"L];
           AskForHelp[FALSE,MyName,0]};
    ENDLOOP;
    IF Termination AND ((TimingStack[TimingStackTop] = CommTime) OR
                        (TimingStack[TimingStackTop] = IdleTime)) THEN
        TimingStack[TimingStackTop] ← TermTime;
    PopTiming[];
  END; – – CheckIncoming

BackTrack: PROCEDURE[] =
< < Recursive backtrack to solve the problem. Uses an explicit stack
    of unfinished work: CurrentProblem. The problem to expand is the
    first in the Heap at the current GivenAwayDepth. > >
BEGIN
  First, Done, Abort: BOOLEAN;
  PrevDepth: INTEGER;
  TmpWork: WorkGivenPtr;
  RootWork: WorkGivenPtr;
  ParentPtr: LONG PDINTER TO ProblemType;
  NeedUpdate. BOOLEAN;
  inMsgPtr: MsgBufferPtr ← NIL;
  Empty: BOOLEAN;

  HaveBeenIdle ← FALSE;
  IF AvailWorkCount > 0 THEN
    {RootWork ← SelectWork[MyName];
     {OPEN RootWork ↑ ;
      CurrentRedundancy ← Redundancy;
      PushTiming[TimeMeasures + CurrentRedundancy];
      CurrentProblem[0] ← Value;
      PrevDepth ← – 1;
      Depth ← 0;
      GiveAwayDepth ← 0;
      DeweyPrefix ← ID};
     IF Debug > = 1 THEN
       {PushTiming[TossTime];
        WriteLongDecimal[Timespent ← ElapsedTime[StartTime]],
        WriteString[" Backtrack on "L];
        PrintDewey[RootWork ↑ .ID];
        NewLine[];
        PopTiming[]};
     DO
       IF PrevDepth < Depth THEN
         {IF Depth = GiveAwayDepth THEN
            {TmpWork ← FindWorkGiven[DeweyPrefix];
             ParentPtr ← @(TmpWork ↑ .Value)}
          ELSE ParentPtr ← @(CurrentProblem[Depth]);
          First ← TRUE;
          IF Debug > = 6 THEN
              {PrintProblem[ParentPtr ↑ ];
               WriteString[" "L];
               PrintAnswer[ParentPtr ↑ ];
               NewLine[]};
          Done ← Generate[First,ParentPtr,@CurrentProblem[Depth + 1]];
          PrevDepth ← Depth;
          IF NOT Done THEN
              {Depth ← Depth + 1;
               DeweyDelta[Depth] ← 1};
          < < ELSE go back up; leave PrevDepth = Depth as a sigh there
                were no children of this node. > > }
        ELSE {IF Depth < = GiveAwayDepth THEN
                {IF Debug > = 2 THEN
                    {PushTiming[TossTime];
                     WriteLongDecimal[Timespent ← ElapsedTime[StartTime]];
                     WriteString[" Just finished "L];
                     PrintDewey[DeweyPrefix];
                     PopTiming[]};
                 TmpWork ← FindWorkGiven[DeweyPrefix];
                 IF Debug > = 1 THEN
                    {WriteString["In backtrack workdone...."L];
                     WriteLongDecimal[TmpWork ↑ .Value.LeafCount];
                     NewLine[]};
                 WorkDone[DeweyPrefix, @TmpWork ↑ .Value];
                 EXIT}
              ELSE { – – Depth > GivenAwayDepth
                     IF Depth = GiveAwayDepth + 1 THEN
                       {TmpWork ← FindWorkGiven[DeweyPrefix];
                        IF TmpWork = NIL THEN EXIT;
                        ParentPtr ← @(TmpWork ↑ .Value)}
                     ELSE {ParentPtr ← @(CurrentProblem[Depth – 1]);
                           TmpWork ← NIL},
                     NeedUpdate ← Combine[@CurrentProblem[Depth], ParentPtr];
                     IF NeedUpdate AND (TmpWork # NIL) THEN
                       DoUpdate[DeweyPrefix, ParentPtr];
                     First ← FALSE;
                     Done ← Generate[First,ParentPtr,@CurrentProblem[Depth]],
                     IF Done THEN
                       {PrevDepth ← Depth;
                        Depth ← Depth – 1;
                        Process.Yield[]}
                     ELSE { – – found a sibling
                            DeweyDelta[Depth] ← DeweyDelta[Depth] + 1
                            PrevDepth ← Depth – 1;
                            Empty ← InQueueEmptyCheck[];
                            IF NOT Empty THEN
                              {Abort ← CheckIncoming[FALSE];
                               IF Termination OR Abort THEN
                                 {IF Abort AND
                                      (RootWork ↑ .Redundancy > CurrentRedundancy)
                                  THEN RootWork ↑ .Redundancy ←
                                           RootWork ↑ .Redundancy – 1
                                  EXIT}}}}}
      ENDLOOP;
      PopTiming[]};
    END; – – BackTrack

Synchronize: PROCEDURE[] =

< <If we are machine PartitionSize, send startup messages to the other
  machine. Otherwise, wait for such a message before continuing. > >

  BEGIN
    Machine: INTEGER ← 1;
    inMsgPtr: MsgBufferPtr ← NIL;
    a: InterNodeMessage Message;

    PushTiming[TossTime];
    IF MyName = PartitionSize THEN
      {WHILE Machine < MyName
          DO
          outMsgPtr ← zone.NEW[MsgBuffer];
          a.Kind ← Synch;
          outMsgPtr ↑ .message ← a;
          outMsgPtr ↑ .source ← MyName;
```

```
            outMsgPtr↑.dest ← Machine;
            Sender[outMsgPtr];
            Machine ← Machine + 1;
          ENDLOOP}
      ELSE – – wait for last machine to awaken us
          {DO
              WHILE inMsgPtr = NIL
                DO
                Process.Pause[30];
                inMsgPtr ← GetInData[];
                ENDLOOP;
              IF inMsgPtr↑.source = PartitionSize THEN
                WITH a: inMsgPtr↑.message SELECT FROM
                    interNodeMessage = > {
                      IF a.Kind = Synch THEN
                        {zone.FREE[@inMsgPtr];
                         EXIT}};
                    ENDCASE;
              zone.FREE[@inMsgPtr];
            ENDLOOP};
        StartTime ← Get[];
        PopTiming[];

    END; – – Synchronize

  NodeDoIt: PROCEDURE[] =
  < <A process gets a larger stack, and we need that. Accept work from
    the host, either 'ask for help' or 'here is a problem' Synchronize
    with the other machines, then do what was asked of us. > >

    BEGIN
      P: ProblemType;
      Style: INTEGER;
      Seq: INTEGER ← 0;
      Dew: Dewey;
      Dummy: BOOLEAN;
      ParPtr. WorkGottenPtr;
      inMsgPtr: MsgBufferPtr ← NIL;

      BEGIN
          ENABLE
            ABORTED = > {IF CurrentProblem # NIL THEN
                          zone.FREE[@CurrentProblem],
                        CONTINUE};

      WHILE MyName = 0
          DO
          Process.Pause[60];
          data ← GetWindowHandle[];
          IF data # NIL THEN
              MyName ← data.nodeNumber;
          ENDLOOP;
        pgen ← Randomize[MyName];
        DO
          IF Debug > = 1 THEN
              {WriteString["Waiting for Message from Host."L];
                NewLine[]; NewLine[]},
          DO – – ignore spurious messages
            inMsgPtr ← GetInData[],
            IF inMsgPtr↑ source = 0 THEN
                EXIT
            ELSE – – not from host
              {IF Debug > = 3 THEN
                  {WriteString["got a spurious message from "L];
                    WriteDecimal[inMsgPtr↑ source];
                    NewLine[]}},
            zone.FREE[@inMsgPtr];
            ENDLOOP;
          WITH a: inMsgPtr↑ message SELECT FROM
              ProblemMessage = > {
                Style ← a.Kind,
                P ← a.Prob;
                AnswerSpec ← a.Answer;
                PartitionSize ← a.PartSize,
                WorkFract ← a.WorkFraction;
                WorkFract ← – 1;
                NumHelpers ← a.NumHelp;
                IF NumHelpers > = PartitionSize THEN
                  NumHelpers ← PartitionSize – 1;
                Debug ← a.DebugLevel};
            ENDCASE;
        zone.FREE[@inMsgPtr];
        WHILE Seq < MyName – 1
          DO
          HelperList[Seq] ← Seq + 1;
          Seq ← Seq + 1
          ENDLOOP,
        WHILE Seq < PartitionSize – 1
          DO
          HelperList[Seq] ← Seq + 2;
          Seq ← Seq + 1
          ENDLOOP;
        CurrentProblem ← zone.NEW[CurrentProblemType];
        Termination ← FALSE;
        NextToAsk ← MyName MOD PartitionSize + 1;
        DeweyPrefix.Length ← MaxDepth,
        StartTime ← Get[];
        ApplicInit[];
        StartTiming[CommTime],
        Synchronize[];
        Seq ← 0;
        WHILE Seq < = 3 + MaxRedundancy
          DO
          Time[Seq] ← 0;
          Seq ← Seq + 1;
          ENDLOOP;
        LocalSequence ← 0,
        IF Style = 0 THEN
          {DEBUG[2, "Start by begging "];
           CurrentRedundancy ← 10;
           Depth ← 0;
           GiveAwayDepth ← 1;
           WHILE (NOT Termination) AND (AvailWorkCount = 0)
             DO
               AskForHelp[TRUE,MyName,0];
               Dummy ← CheckIncoming[TRUE];
             ENDLOOP}
        ELSE – – we are born rich
          {CurrentRedundancy ← 0;
           Dew.Length ← 0;
           ParPtr ← Getting[Dew,P,0];
           Giving[ParPtr,P,Dew,Open,CurrentRedundancy]};
        WHILE (NOT Termination) AND (AvailWorkCount # 0)
          DO
          BackTrack[];
          Depth ← 0;
          GiveAwayDepth ← 1;
          IF (NOT Termination) AND (AvailWorkCount = 0) THEN
            {AskForHelp[FALSE,MyName,0];
             CurrentRedundancy ← 10,
             WHILE (NOT Termination) AND (AvailWorkCount = 0)
               DO
               Dummy ← CheckIncoming[TRUE];
               ENDLOOP};
          < < ELSE either (1) we aborted or (2) finished some work or
                      (3) heard Termination. In cases (1) and (2), there
                      is already something waiting of greater importance
                      in WorkGiven > >
          ENDLODP,
          IF Termination THEN
            DropOut[];
          zone.FREE[@CurrentProblem];
          HouseKeeping[];
        ENDLOOP;
        END;
    END; – – NodeDoIt
```

```
    – – main
    Process.Detach[FORK NodeDoIt[]];
END. – – NodeImpl
```

```
DIRECTORY
    IO,
    TimeMeasureDfs,
    System USING[GetClockPulses, GetGreenwichMeanTime, PulsesToMicroseconds];

TimeMeasureImpl: PROGRAM
    IMPORTS IO, System
    EXPORTS TimeMeasureDfs =

    BEGIN
        OPEN IO, System, TimeMeasureDfs;

        Get: PUBLIC PROCEDURE[] RETURNS [TimeMeasure] =
            {RETURN[TimeMeasure[gmt: GetGreenwichMeanTime[],
                    pulses: GetClockPulses[]]]};

        ElapsedTime: PUBLIC PROCEDURE[time: TimeMeasure] RETURNS[Millisecs] =
            BEGIN
                timeNew: TimeMeasure ← Get[];
                RETURN[
                    IF timeNew.gmt – time.gmt > 3600 THEN (timeNew.gmt – time.gmt)*1(
                    ELSE PulsesToMicroseconds[[timeNew.pulses – time.pulses]]/1000];
            END;

        WriteMillisecs: PUBLIC PROC [m: Millisecs] =
            BEGIN
                sec: LONG CARDINAL;
                min: LONG CARDINAL;
                hr: LONG CARDINAL;

                sec ← m/1000;
                min ← sec/60;
                hr ← min/60;
                m ← m MOD 1000;
                sec ← sec MOD 60;
                min ← min MOD 60;
                WriteLongDecimal[hr]; WriteChar[':];
                WriteLongDecimal[min]; WriteChar[':];
                WriteLongDecimal[sec]; WriteChar['.];
                WriteLongDecimal[m];
            END;

    END.
```